TUGAS AKHIR

IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING*

Sulistyawan Abdillah Rosyid
NIM. 11181079

Nisa Rizqiya Fadhliana, S.Kom., M.T.
Rizky Amelia, S.Si., M.Han.

Program Studi Informatika
Jurusan Sains dan Analitika Data
Fakultas Sains dan Teknologi Informasi
Institut Teknologi Kalimantan
Balikpapan, 2025

TUGAS AKHIR

IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING*

Sulistyawan Abdillah Rosyid
NIM. 11181079

Nisa Rizqiya Fadhliana, S.Kom., M.T.
Rizky Amelia, S.Si., M.Han.

Program Studi Informatika
Jurusan Sains dan Analitika Data
Fakultas Sains dan Teknologi Informasi
Institut Teknologi Kalimantan
Balikpapan, 2025

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul **IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING***
adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri. Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Balikpapan, 10 Juni 2025

Sulistyawan Abdillah
Rosyid
NIM. 11181079

(Halaman ini sengaja dikosongkan)

PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR
UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Institut Teknologi Kalimantan, saya yang bertanda tangan di bawah ini :

Nama	: Sulistyawan Abdillah Rosyid
NIM	: 11181079
Program Studi	: Informatika
Jurusan	: Jurusan Sains dan Analitika Data

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Institut Teknologi Kalimantan Hak Bebas Royalti Non-eksklusif (Non-exclusive Royalty Free Right) atas karya ilmiah saya yang berjudul :

IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING*

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini, Instutut Teknologi Kalimantan berhak menyimpan, mengalihmediakan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Balikpapan, 10 Juni 2025

Sulistyawan Abdillah Rosyid

NIM. 11181079

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

TUGAS AKHIR

Disusun untuk memenuhi syarat memperoleh gelar

Sarjana S.Kom

Pada

Program Studi S-1 Informatika

Jurusan Sains dan Analitika Data

Fakultas Sains dan Teknologi Informasi

Institut Teknologi Kalimantan

Judul Tugas Akhir :

IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING*

Oleh:

Sulistyawan Abdillah Rosyid

11181079

Disetujui oleh Tim Penguji Tugas Akhir :

1.	Nisa Rizqiya Fadhliana, S.Kom., M.T.	Pembimbing I	.....
2.	Rizky Amelia, S.Si., M.Han.	Pembimbing II	.....
3.	Ramadhan Paninggalih, S.Si., M.Si., M.Sc.	Penguji I	.....
4.	Bowo Nugroho, S.Kom., M.Eng.	Penguji II	.....

BALIKPAPAN
JUNI, 2025

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas berkat dan anugerah-Nya sehingga kami dapat menyelesaikan proposal tugas akhir yang berjudul:

"IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING*"

Proposal tugas akhir ini merupakan salah satu syarat yang harus ditempuh untuk menyelesaikan Program Sarjana di Program Studi Informatika, Jurusan Matematika Teknologi dan Informasi, Institut Teknologi Kalimantan (ITK) Balikpapan. Untuk itu penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Tuhan YME atas segala nikmat yang diberikan sehingga dapat terlaksana pembuatan laporan ini.
2. Ibu Nisa Rizqiya Fadhliana, S.Kom., M.T. selaku Dosen Pembimbing Utama dan Koordinator Program Studi Informatika Jurusan Matematika Teknologi dan Informasi ITK.
3. Ibu Rizky Amelia, S.Si., M.Han. selaku Dosen Pembimbing Pendamping dan Dosen Wali dari penulis.
4. Bapak Ibu Seluruh Dosen serta Tenaga Kependidikan Program Studi Informatika Jurusan Matematika dan Teknologi Informasi ITK.
5. Khoirun Nisa Al Fahmi selaku istri dari penulis dan keluarga dari penulis yang selalu suportif untuk menyelesaikan laporan ini.
6. Serta semua pihak yang terlibat dalam penyusunan proposal tugas akhir ini.

Penulis menyadari bahwa penyusunan laporan Tugas Akhir ini masih jauh dari sempurna, karena itu kami mengharapkan segala kritik dan saran yang membangun. Semoga Tugas Akhir ini dapat bermanfaat bagi kita semua. Atas perhatiannya penulis ucapkan terima kasih.

Balikpapan, 10 September 2024

Penyusun

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN *SNAKE* DENGAN METODE *REINFORCEMENT LEARNING*

Nama Mahasiswa	: Sulistyawan Abdillah Rosyid
NIM	: 11181079
Dosen Pembimbing Utama	: Nisa Rizqiya Fadhliana, S.Kom., M.T.
Pembimbing Pendamping	: Rizky Amelia, S.Si., M.Han.

ABSTRAK

Penelitian ini bertujuan untuk mengimplementasikan algoritma Reinforcement Learning (RL) berbasis Deep Q-Network (DQN) pada permainan Snake guna mengembangkan agen kecerdasan buatan (AI) yang mampu belajar dan bermain secara optimal. Lingkungan simulasi dikembangkan menggunakan PyGame, dan pelatihan dilakukan selama 5000 episode dengan dua pendekatan: Bellman dan Monte Carlo. Hasil pelatihan menunjukkan bahwa agen dengan pendekatan Bellman mencapai skor rata-rata sebesar 23,58 dan panjang ular maksimum sebesar 70 unit, dengan peningkatan performa signifikan dalam 1000 episode pertama sebelum mengalami stagnasi. Sementara itu, model Monte Carlo menunjukkan peningkatan yang lebih lambat namun stabil setelah 500 episode, dengan skor rata-rata berkisar antara 21–22 dan panjang ular maksimum sekitar 67 unit. Evaluasi dilakukan berdasarkan metrik performa seperti skor rata-rata, panjang maksimum, dan stabilitas reward antar episode, serta dilengkapi dengan observasi perilaku agen yang menunjukkan kemampuan dalam menghindari dinding, tubuh sendiri, dan keluar dari situasi jebakan. Temuan ini menunjukkan bahwa algoritma DQN efektif dalam membentuk agen yang adaptif dan responsif terhadap lingkungan dinamis.

Kata Kunci: *Agent, Artificial Intelligence, Deep Q-Network, Reinforcement Learning*

(Halaman ini sengaja dikosongkan)

IMPLEMENTATION OF ARTIFICIAL INTELLIGENCE IN SNAKE GAME WITH REINFORCEMENT LEARNING METHOD

By	: Sulistyawan Abdillah Rosyid
Student Identity Number	: 11181079
Supervisor	: Nisa Rizqiya Fadhliana, S.Kom., M.T.
Co-Supervisor	: Rizky Amelia, S.Si., M.Han.

ABSTRACT

This research aims to implement a Reinforcement Learning (RL) algorithm using the Deep Q-Network (DQN) approach in the Snake game to develop an artificial intelligence (AI) agent capable of learning and playing optimally. The simulation environment was built using PyGame, and the training process was conducted over 5000 episodes using two approaches: Bellman and Monte Carlo. The results show that the agent trained with the Bellman approach achieved an average score of 23.58 and a maximum snake length of 70 units, with a significant performance increase in the first 1000 episodes before reaching a stagnation point. Meanwhile, the Monte Carlo model demonstrated slower but more stable performance growth after 500 episodes, achieving an average score between 21 and 22 and a maximum snake length of approximately 67 units. The evaluation was based on key performance metrics such as average score, maximum length, and reward stability per episode, supported by behavioral observations indicating that the agent learned to avoid walls, its own body, and escape from self-created traps. These findings indicate that the DQN algorithm is effective in developing an adaptive and responsive agent in a dynamic environment

Keywords: Agent, Artificial Intelligence, Deep Q-Network, Reinforcement Learning

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

Penelitian ini berfokus pada penerapan *Artificial Intelligence* (AI) dalam permainan Snake dengan menggunakan metode *Reinforcement Learning*. Permainan Snake dipilih karena strukturnya yang sederhana namun cukup kompleks untuk dijadikan lingkungan pembelajaran bagi agen cerdas. Penelitian ini bertujuan untuk menguji efektivitas metode tersebut dalam mengembangkan agen yang adaptif, serta memberikan gambaran mengenai potensi penerapan AI pada gim klasik sebagai media pembelajaran dan eksperimen algoritma pembelajaran mesin.

Pada bab ini menjelaskan mengenai pendahuluan dari penelitian yang terdiri dari latar belakang, rumusan masalah, tujuan, manfaat dari dilakukannya penelitian ini.

1.1 Latar Belakang

Artificial intelligence (AI) merupakan sebuah istilah yang sering digaungkan akhir-akhir ini. Salah satu penyebabnya adalah saat ChatGPT dirilis oleh OpenAI pada November 2022. Sejak saat itu banyak produk-produk AI lain mulai bermunculan. Metode *Deep Learning* (DL), *Natural Language Processing* (NLP), *Computer Vision* banyak digunakan oleh perusahan penghasil AI. AI sendiri sebenarnya bukanlah sesuatu yang baru. Gagasan mengenai AI ini ditemukan pada tahun 1956, tetapi karena keterbatasan kala itu membuat gagasan AI ini menjadi sesuatu yang diremehkan oleh sebagian besar orang. Saat ini AI dibuat dengan tiga cara yaitu *Supervised Learning, Unsupervised Learning*, dan *Reinforcement Learning*.

Perkembangan teknologi komputer dan *Artificial Intelligence* (AI) telah mengalami kemajuan yang pesat dalam beberapa tahun terakhir. Salah satu cabang AI yang mendapatkan perhatian khusus adalah cabang *Machine Learning* (ML). Cabang ini mencakup beragam metode dan algoritma yang mengajarkan komputer

untuk cara untuk mengambil keputusan dan tindakan. Dari beragam model *Machine Learning, Reinforcement Learning* (RL) cukup menonjol sebagai teknik yang menjanjikan untuk aplikasi di lingkungan yang dinamis dan kompleks. Dalam model ini sebuah agen belajar melalui interaksinya dengan lingkungan untuk memaksimalkan sebuah *reward*.

Permainan "*Snake*" adalah sebuah gim klasik yang ideal untuk mengembangkan algoritma *Reinforcement Learning* (RL) karena kesederhanaan dan tantangan pada gim ini. Dalam gim ini, agen (*Snake*) diharuskan untuk mencari dan mendapatkan makanan yang muncul secara acak sembari menghindari agar tidak tertabrak dinding atau diri agen sendiri yang selalu bertumbuh setiap memakan sebuah makanan. Penerapan AI pada gim *Snake* menggunakan teknik RL memberikan beberapa kemungkinan, termasuk sebagai platform yang dapat diskalakan untuk menguji algoritma *Reinforcement Learning* dalam lingkungan yang terkendali dan dinamis. Aplikasi ini juga dapat disesuaikan dengan bidang lainnya seperti robotika dan navigasi. Gim ini memungkinkan proses pembelajaran dan adaptasi agen AI divisualisasikan dengan jelas. Hal ini memberikan wawasan penting tentang bagaimana algoritma *Reinforcement Learning* belajar dari pengalaman.

Pada jurnal sebelumnya milik Crespo dan Wichert (2020), dalam pembelajaran penguatan *Reinforcement Learning*, *Deep Q-Network*s (*DQN*) digunakan secara luas untuk mempelajari keputusan optimal dalam lingkungan permainan, seperti pada gim Atari. *DQN* memanfaatkan jaringan saraf untuk mendekati fungsi nilai yang memprediksi seberapa baik suatu tindakan dalam situasi tertentu. Peningkatan penting pada metode *DQN* meliputi *Averaged-DQN*, yang menstabilkan proses pembelajaran dengan mengurangi variabilitas dalam perkiraan *Q-values*, serta *Rainbow DQN*, yang menggabungkan beberapa perbaikan, termasuk *Double Deep Q-Learning*, *dueling networks*, dan *prioritized replay* untuk meningkatkan performa algoritma dalam berbagai tugas gim (Crespo dan Wichert, 2020).

Penelitian ini bertujuan untuk mengembangkan agen yang mampu bermain *Snake* dengan sangat baik dan mengevaluasi efisiensi dan efektivitas berbagai algoritma RL yang diterapkan untuk permainan *Snake*. Selain itu, penelitian ini juga

bertujuan untuk menemukan masalah dan peluang dalam penerapan RL dalam permainan *Snake* dan memberikan saran untuk penelitian tambahan. Penelitian ini diharapkan dapat meningkatkan pemahaman kita tentang *Reinforcement Learning* secara teoritis dan dalam aplikasinya yang kompleks dan dinamis. Ini juga akan membuka jalan untuk inovasi lebih lanjut dalam penggunaan AI untuk aplikasi yang kompleks dan dinamis.

1.2 Perumusan Masalah

Berdasarkan latar belakang, maka didapatkan rumusan masalah sebagai berikut:

1. Bagaimana mengimplementasikan *Reinforcement Learning* untuk membuat sebuah agen AI yang mampu bermain gim "*Snake*" secara baik ?
2. Apa saja parameter yang memengaruhi kinerja agen dalam permainan dan bagaimana pengaruhnya ?
3. Bagaimana mengukur efektivitas algoritma *Reinforcement Learning* dengan metode Bellman dibandingkan dengan metode lain dalam permainan Snake?

1.3 Tujuan Penelitian

Sesuai dengan rumusan masalah, tujuan dari penelitian ini adalah sebagai berikut:

1. Mengimplementasikan algoritma *Reinforcement Learning* pada sebuah agen untuk gim "*Snake*" secara baik.
2. Mengetahui parameter yang memengaruhi kinerja agen dalam permainan dan bagaimana pengaruh dari parameter tersebut terhadap kinerja agen.
3. Membandingkan efektivitas dari algoritma *Reinforcement Learning* pada gim "*Snake*" dengan *framework* Bellman dan Monte Carlo.

1.4 Batasan Penelitian

Adapun batasan masalah pada penelitian adalah sebagai berikut:

1. Algoritma yang digunakan adalah variasi dari jenis-jenis tertentu dari *Reinforcement Learning Deep Q-Network* atau varian lain yang relevan.
2. Implementasi akan dilakukan dengan *framework* atau *library* tertentu seperti PyTorch.
3. Pelatihan agen akan difokuskan pada permainan *Snake* 2 dimensi tanpa modifikasi aturan atau lingkungan permainan yang kompleks.
4. Analisis berfokus pada bagaimana variasi parameter ini mempengaruhi kinerja agen dalam hal skor rata-rata, panjang antrean, dan waktu bertahan hidup.
5. Parameter yang akan dianalisis meliputi ukuran *grid*, kecepatan permainan, struktur *reward system*, dan parameter internal algoritma *Reinforcement Learning* seperti *learning rate*, *discount factor*, dan *exploration rate*.
6. Kinerja agen yang diperoleh dari algoritma *Reinforcement Learning* dilakukan dengan pendekatan metode Bellman dan Monte Carlo.
7. Evaluasi didasarkan pada metrik kinerja seperti skor rata-rata yang dicapai oleh agen, panjang antrean maksimum yang dicapai, dan stabilitas kinerja.
8. Penelitian ini dibatasi pada evaluasi dalam lingkungan simulasi, tanpa penerapan langsung di luar lingkup permainan *Snake*.

1.5 Manfaat Penelitian

Adapun manfaat secara umum dari penelitian ini adalah memberikan wawasan lebih dalam tentang cara kerja algoritma pembelajaran penguatan dalam lingkungan dinamis berbasis aturan seperti Permainan *Snake*. Hasil penelitian ini dapat mengarah pada pengembangan dan peningkatan algoritma *Reinforcement Learning* yang lebih efisien dan efektif. Permainan Ular berfungsi sebagai studi kasus yang sangat baik untuk membuat prototipe aplikasi RL sebelum menerapkannya pada masalah dunia nyata yang lebih kompleks.

Peneliti yang berpartisipasi dalam penelitian ini akan mendapatkan pengalaman langsung menerapkan dan menguji algoritma AI serta mengembangkan keterampilan analisis data dan pemecahan masalah. Studi ini memungkinkan peneliti untuk mengembangkan keterampilan teknis dalam pemrograman, menggunakan *libary* AI seperti TensorFlow dan PyTorch, dan

menerapkan metode ilmiah. Selain itu, teknik dan wawasan yang diperoleh dengan menerapkan *Reinforcement Learning* pada gim *Snake* dapat diterapkan pada berbagai bidang lainnya, seperti robotika, navigasi otonom, dan pengelolaan sumber daya.

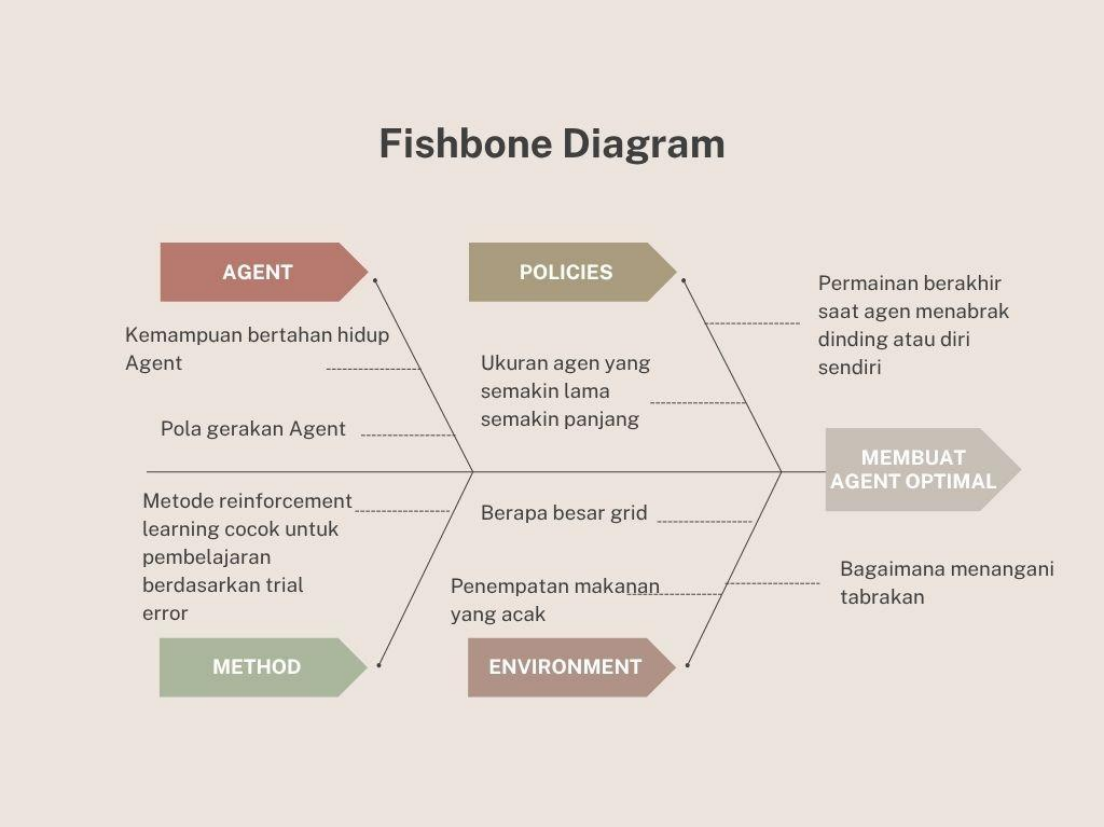
Penelitian ini juga memiliki manfaat signifikan bagi industri permainan. Dengan memahami dan menerapkan algoritma RL yang lebih baik, pengembang permainan dapat menciptakan pengalaman bermain yang lebih menarik dan adaptif. Algoritma yang lebih efisien dapat digunakan untuk mengembangkan karakter non-pemain (NPC) yang lebih cerdas, meningkatkan interaksi pemain, dan menciptakan tantangan yang lebih dinamis dalam permainan. Selain itu, teknik yang diperoleh dari penelitian ini dapat membantu dalam pengembangan sistem rekomendasi yang lebih baik, yang dapat meningkatkan keterlibatan pemain dan retensi pengguna dalam permainan.

1.6 Kerangka Pemikiran

Dalam upaya mengembangkan agen kecerdasan buatan yang mampu bermain permainan Snake secara optimal, diperlukan suatu kerangka pemikiran yang sistematis. Permainan Snake dipilih karena meskipun tampak sederhana, ia menawarkan kompleksitas yang cukup tinggi dalam hal pengambilan keputusan secara real-time, adaptasi terhadap kondisi lingkungan, serta evaluasi strategi berdasarkan reward. Oleh karena itu, metode Reinforcement Learning (RL), khususnya pendekatan Deep Q-Network (DQN), dianggap paling tepat karena memungkinkan agen belajar dari pengalaman melalui proses trial-and-error untuk meningkatkan kinerja dari waktu ke waktu.

Kerangka pemikiran dalam penelitian ini berfokus pada hubungan antara berbagai komponen kunci yang memengaruhi kinerja agen AI, seperti algoritma yang digunakan, konfigurasi lingkungan permainan, parameter pelatihan, dan evaluasi performa. Fishbone diagram yang digunakan memvisualisasikan bagaimana faktor-faktor seperti kemampuan agen, aturan permainan, metode pembelajaran, dan kondisi simulasi bekerja secara sinergis untuk mencapai tujuan akhir, yaitu menciptakan agen AI yang tidak hanya mampu bertahan hidup dalam

permainan, tetapi juga mampu memaksimalkan reward secara konsisten. Dengan pendekatan ini, penelitian diharapkan mampu memberikan gambaran menyeluruh terhadap proses pengembangan agen AI berbasis RL dalam lingkungan permainan yang dinamis dan penuh tantangan



Gambar 1. 1 Fishbone Diagram

Fishbone yang digambarkan oleh gambar 1.1 memiliki berbagai faktor yang memengaruhi keberhasilan dalam menciptakan agen kecerdasan buatan (AI) yang optimal untuk permainan *Snake* menggunakan pendekatan *Reinforcement Learning*. Faktor pertama berasal dari sisi *Agent*, di mana kemampuan agen untuk bertahan hidup serta pola gerakannya menjadi aspek krusial dalam menentukan seberapa baik agen tersebut dapat merespons perubahan lingkungan dalam permainan. Agen yang memiliki strategi gerakan adaptif cenderung memiliki peluang lebih besar untuk bertahan lama dan memperoleh skor lebih tinggi.

Selanjutnya, dari aspek *Policies*, terdapat aturan permainan yang harus diikuti agen, seperti bertambahnya panjang tubuh ular seiring waktu serta kondisi

kekalahan ketika agen menabrak dinding atau tubuhnya sendiri. Aturan-aturan ini menambah kompleksitas tantangan yang harus dihadapi oleh agen dalam setiap langkah pengambilan keputusan.

Kemudian dari sisi *Method*, *Reinforcement Learning* dianggap metode yang tepat karena sesuai untuk skenario "*trial and error"*, memungkinkan agen belajar dari setiap kegagalan dan perolehan reward secara langsung selama proses bermain.

Faktor *Environment* juga memberikan pengaruh signifikan terhadap kinerja agen. Permainan Snake menyajikan lingkungan yang berubah-ubah, misalnya ukuran grid yang bervariasi dan penempatan makanan yang dilakukan secara acak. Hal ini mengharuskan agen untuk mampu menyesuaikan strategi dengan kondisi yang tidak sepenuhnya dapat diprediksi. Lingkungan tersebut juga menuntut adanya mekanisme yang efisien dalam menangani situasi benturan, baik dengan dinding maupun dengan tubuh agen sendiri. Semua faktor ini saling berkaitan dan membentuk sistem pembelajaran yang kompleks namun terukur, dengan tujuan akhir untuk menciptakan agen AI yang cerdas, adaptif, dan optimal dalam menjalankan tugasnya di dalam permainan Snake

..

BAB 2
TINJAUAN PUSTAKA

Pada bab ini akan menjelaskan terkait tinjauan pustaka yang digunakan pada penelitian.

2.1 *Artificial intelligence* (AI)

Artificial intelligence (AI) adalah cabang ilmu komputer yang berfokus pada pengembangan sistem yang mampu melakukan tugas yang biasanya membutuhkan kecerdasan manusia, seperti pengenalan suara, pengambilan keputusan, dan terjemahan bahasa, AI didefinisikan sebagai studi tentang agen cerdas, yaitu entitas yang dapat merasakan lingkungan mereka dan mengambil tindakan untuk memaksimalkan peluang keberhasilan dalam mencapai tujuan tertentu. Dalam definisi ini, AI mencakup berbagai pendekatan, mulai dari logika simbolik hingga pembelajaran mesin yang lebih modern (Russell, 2020).

Dalam konteks pembelajaran mesin, melibatkan algoritma yang memungkinkan komputer belajar dari data dan memperbaiki performa mereka dari waktu ke waktu tanpa diprogram secara eksplisit. Pendekatan ini telah berkembang pesat dalam beberapa tahun terakhir dan diterapkan dalam berbagai bidang, termasuk kesehatan, transportasi, dan keuangan, untuk membuat prediksi yang lebih akurat dan otomatisasi proses yang kompleks (Goodfellow, dkk., 2016).

AI dapat dikelompokkan menjadi beberapa kategori utama berdasarkan kemampuannya, yaitu AI lemah (narrow AI) dan AI kuat (general AI). AI lemah dirancang untuk melakukan tugas tertentu dengan sangat baik, seperti pengenalan wajah atau pengolahan bahasa alami, tetapi tidak memiliki kemampuan untuk melakukan tugas di luar dari spesialisasinya. Sementara itu, AI kuat adalah sebuah konsep di mana mesin memiliki kecerdasan umum yang setara dengan manusia, dengan kemampuan untuk belajar dan beradaptasi dalam berbagai situasi. Meskipun AI kuat masih merupakan tujuan jangka panjang yang belum tercapai, AI lemah sudah diterapkan luas dalam kehidupan sehari-hari (Tegmark, 2017).

Perkembangan AI yang pesat membawa potensi besar tetapi juga tantangan serius terkait etika, keamanan, dan dampak sosial. Bostrom mengemukakan bahwa ketika AI mencapai tingkat super intelligence, atau kecerdasan yang jauh melampaui kecerdasan manusia, ini dapat membawa perubahan yang sangat besar dalam peradaban manusia, termasuk peluang dan ancaman eksistensial. Oleh karena itu, Bostrom menekankan pentingnya penelitian untuk memastikan bahwa AI berkembang dengan cara yang aman dan bermanfaat bagi umat manusia (Bostrom, 2014).

2.2 Machine Learning

Belakangan ini Pembelajaran Mesin / *Machine Learning* sungguh banyak digunakan, bahkan bisa lebih banyak dari yang kita perkirakan. Contohnya saat kita mencari "Hadiah" untuk teman atau sahabat kita di internet. Kita akan mengetikkan sebuah kata kunci maka muncullah hal-hal yang berkaitan dengan kata kunci tersebut. Saat kita membuka kotak email kita lagi, tanpa disadari, kita akan menemukan sebuah email sejenis masuk ke kotak email kita atau mungkin dapat kita temukan pada *spam*. Selanjutnya saat pergi ke sebuah supermarket untuk membeli popok untuk dijadikan hadiah kepada sahabat. Saat ingin membayar, kasir memberikan sebuah kupon diskon untuk bir. Aksi kasir memberikan sebuah kupon diskon pada orang yang membeli popok ditunjukkan oleh *software* juga akan membeli bir. Data itulah yang dapat digunakan untuk mencari strategi untuk bisnis.

Apa yang akan kita cari pada sebuah data mentah itu tidak akan jelas. Contohnya dalam mendeteksi sebuah email *spam* mencari sebuah kata dalam email tentu tidak akan terlalu membantu. Bahkan akan terjadi kesalahan-kesalahan pendeteksian *email spam*. Karenanya akan lebih baik jika mencari gabungan dari beberapa kata yang ada dalam sebuah *email*, digabungkan dengan panjang *email*, laporan *spam* dari user lain, dan banyak hal lainnya. Dengan pertimbangan itu semua akan menjadi jelas sebuah *email* adalah *spam* atau tidak. Inilah salah satu kegunaan *Machine Learning* untuk mengubah data mentah menjadi informasi yang dapat digunakan (Harrington, 2012).

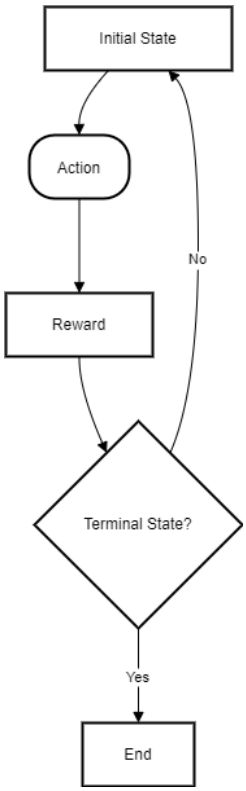
Machine Learning adalah cabang dari kecerdasan buatan (*artificial intelligence*) yang berfokus pada pengembangan algoritma dan teknik yang memungkinkan komputer untuk belajar dari data. Dalam pengertian ini, mesin dapat belajar dari pengalaman atau data masa lalu untuk membuat prediksi atau keputusan tanpa harus diprogram secara eksplisit untuk setiap tugas (Mitchell, 1997).

Machine Learning melibatkan proses di mana model atau algoritma "dilatih" dengan menggunakan data historis. Setelah proses pelatihan ini, model dapat digunakan untuk membuat prediksi atau menyelesaikan masalah tertentu berdasarkan data baru. Proses ini mencakup berbagai teknik, seperti *supervised learning*, *unsupervised learning*, dan *Reinforcement Learning,* yang masing-masing memiliki aplikasi dan pendekatan yang berbeda tergantung pada jenis data dan tujuan analisis (Alpaydin, 2020).

Machine Learning telah menjadi landasan utama dalam berbagai bidang, mulai dari pengenalan suara dan gambar hingga analisis prediktif dalam bisnis dan industri. Machine Learning memungkinkan sistem untuk mengenali pola kompleks dalam data dan secara otomatis memperbaiki kinerjanya seiring waktu, membuatnya menjadi alat yang sangat kuat dalam era data besar saat ini (Goodfellow, dkk., 2016)

2.3 Reinforcement Learning

Reinforcement Learning (RL) adalah suatu metode pembelajaran mesin di mana sebuah agen belajar untuk membuat keputusan dengan melakukan aksi-aksi di dalam lingkungan tertentu guna memaksimalkan suatu nilai *reward* kumulatif. *Reinforcement Learning* menekankan pada interaksi antara agen dan lingkungan melalui percobaan dan kesalahan, di mana agen tersebut mengamati status lingkungannya, memilih aksi, dan kemudian menerima *feedback* berupa *reward*. Proses ini memungkinkan agen untuk belajar secara mandiri tanpa memerlukan data pelatihan yang lengkap seperti dalam *supervised learning* (Sutton dan Barto, 2018).



Gambar 2. 1 Skema *Reinforcement Learning*

Dalam konteks ini, agen bertujuan untuk menemukan kebijakan (*policy*) yang menentukan bagaimana agen harus bertindak di berbagai situasi agar mendapatkan *reward* maksimal. *Reinforcement Learning* sering kali digunakan dalam skenario di mana keputusan harus dibuat secara berurutan dan dampaknya mungkin tidak segera terlihat, seperti dalam permainan video gim atau pengendalian robotika (Mnih, dkk., 2015).

Selain itu, *Reinforcement Learning* memiliki berbagai aplikasi di dunia nyata yang semakin berkembang. *Reinforcement Learning* telah digunakan untuk mengalahkan juara dunia dalam permainan Go, melalui sistem bernama AlphaGo. Sistem ini memanfaatkan teknik *Monte Carlo Tree Search* (MCTS) bersama dengan *deep neural networks* untuk mengevaluasi posisi dan memilih langkah terbaik. Keberhasilan ini menyoroti potensi RL dalam menangani masalah kompleks yang membutuhkan pengambilan keputusan strategis di berbagai

domain, mulai dari permainan hingga industri otomotif dan keuangan (Silver, dkk., 2016).

Penggunaan *Reinforcement Learning* juga mulai terlihat dalam pengoptimalan proses bisnis dan industri. Pada sektor energi, *Reinforcement Learning* digunakan untuk mengoptimalkan pengelolaan jaringan listrik pintar (smart grid) dan pengaturan sumber daya energi terbarukan. Agen *Reinforcement Learning* dapat dilatih untuk merespon perubahan permintaan dan penawaran listrik, serta memprediksi konsumsi energi untuk meningkatkan efisiensi dan stabilitas sistem (François-Lavet, dkk., 2018).

Secara umum, *Reinforcement Learning* terus berkembang seiring dengan peningkatan kemampuan komputasi dan kemajuan algoritma pembelajaran. Tantangan utama dalam *Reinforcement Learning* meliputi eksplorasi yang efisien, generalisasi terhadap situasi baru, dan pengurangan kompleksitas komputasi. Namun, dengan perkembangan teknik seperti *Deep Reinforcement Learning*, multi-agent *Reinforcement Learning*, dan *Meta Reinforcement Learning*, pendekatan ini memiliki potensi yang besar untuk diterapkan dalam berbagai disiplin ilmu dan industri (Li, 2018).

2.3.1. Q-Learning

Q-Learning bertujuan untuk menemukan kebijakan optimal yang memaksimalkan total *reward* jangka panjang. Pada dasarnya, *Q-Learning* menggunakan fungsi nilai aksi yang dikenal sebagai fungsi Q untuk mengestimasi seberapa baik tindakan tertentu yang dilakukan dalam suatu keadaan tertentu (Watkins, 1992).

Fungsi Q didefinisikan sebagai:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma\, max_{a'}\, Q(s',a') - Q(s,a)) \quad \text{.....(2.1)}$$

Keterangan:

$Q(s,a)$	: Nilai Q
s	: State / Keadaan

a	: Aksi
α	: *Learning Rate* / laju pembelajaran
r	: *Reward*
γ	: *Discount factor*
s'	: *State* Selanjutnya
a'	: Aksi selanjutnya
$max_{a'} Q(s', a')$	: Estimasi nilai maksimum Q untuk semua aksi (a') pada keadaan (s′)

Nilai fungsi Q yang mewakili estimasi nilai dari suatu aksi a yang diambil pada keadaan s. *Learning rate* (laju pembelajaran), yaitu seberapa besar kita memperbarui nilai Q berdasarkan informasi baru. Nilai α biasanya berkisar antara 0 dan 1, di mana nilai yang lebih besar berarti pembelajaran lebih cepat, tetapi mungkin menyebabkan ketidakstabilan jika terlalu besar. *Reward* (hadiah) yang diterima setelah melakukan aksi a di keadaan s, dan pindah ke keadaan berikutnya s′. Ini adalah hadiah langsung yang diperoleh dari lingkungan. *Discount factor* (faktor diskon) menentukan seberapa besar pengaruh *reward* masa depan terhadap nilai Q saat ini. Nilai γ juga berkisar antara 0 dan 1, di mana 1 berarti *reward* masa depan dianggap sepenuhnya, dan 0 berarti hanya *reward* saat ini yang diperhitungkan. Next state (keadaan berikutnya) yang dicapai setelah mengambil aksi a dari keadaan s. Estimasi nilai maksimum Q untuk semua aksi a′ yang bisa diambil pada keadaan berikutnya s′. Maksudnya, dari semua aksi yang mungkin, kita memilih aksi terbaik yang memaksimalkan nilai Q di keadaan s′ (Sutton & Barto, 2018).

2.4 Teori Permainan

Teori permainan adalah cabang matematika yang mempelajari interaksi strategis antara agen rasional, di mana hasil yang dicapai tergantung pada keputusan

yang diambil oleh semua pemain. Teori ini umumnya diterapkan pada situasi kompetitif dan kooperatif yang melibatkan pilihan strategis dari berbagai pemain. Dalam konteks permainan *Snake*, kita dapat menggunakan beberapa prinsip dasar dari teori permainan untuk menganalisis pola perilaku dan strategi optimal yang memungkinkan pemain untuk mencapai hasil terbaik.

Permainan *Snake* adalah permainan pemain tunggal di mana pemain mengendalikan ular yang bergerak di dalam arena untuk mengambil / memakan buah yang muncul di layar. Setiap kali ular memakan buah, panjang tubuhnya bertambah, sehingga membuat permainan lebih sulit karena pergerakan menjadi lebih terbatas. Pemain akan kalah jika ular menabrak dirinya sendiri atau dinding.

Dari perspektif teori permainan, *Snake* dapat dianggap sebagai permainan sekuensial deterministik, di mana setiap langkah pemain berpengaruh terhadap posisi masa depan ular, mengingat panjang tubuh ular yang bertambah dari waktu ke waktu. Dalam teori permainan, ini mirip dengan extensive form games, di mana pemain membuat keputusan secara bertahap dengan informasi lengkap tentang kondisi permainan pada setiap langkah.

Pada permainan *Snake*, pendekatan umum yang sering digunakan pemain adalah strategi serakah (*greedy strategy*), di mana pemain berusaha segera menuju buah yang tersedia dalam langkah tersingkat. Meskipun strategi ini efisien untuk tahap awal, permainan menjadi lebih kompleks seiring dengan bertambahnya panjang ular. Dalam teori permainan, strategi serakah tidak selalu optimal karena bisa menyebabkan kebuntuan di masa depan. Permainan *Snake* juga dapat dilihat sebagai permainan yang mengandalkan heuristik, yaitu aturan-aturan yang digunakan pemain untuk membuat keputusan cepat. Contoh heuristik dalam *Snake* adalah selalu menjaga agar ular bergerak dalam pola melingkar atau persegi agar tetap memiliki ruang gerak yang cukup. Dalam hal ini, teori permainan berbicara tentang bagaimana heuristik membantu dalam membuat keputusan optimal di bawah tekanan waktu dan kondisi yang terbatas.

Dari perspektif ilmu komputer dan teori permainan, strategi optimal dalam *Snake* sering kali dikembangkan melalui pendekatan algoritmis. Beberapa algoritma yang relevan dengan analisis teori permainan seperti algoritma *Djikstra* (*A-star*) dan *Hamiltonian Cycle*.

Dalam permainan ini, *reward* ular (agen) dalam memperoleh buah berbanding terbalik dengan risiko bertambahnya panjang tubuh yang dapat menyebabkan kekalahan. Setiap kali pemain memperoleh keuntungan (buah), tantangan juga meningkat, sehingga menghasilkan keseimbangan antara risiko dan *reward*, sebuah konsep yang umum dalam teori permainan. Teori permainan sering kali melibatkan manajemen risiko, di mana pemain harus memilih antara strategi berisiko tinggi dengan *reward* besar atau strategi konservatif dengan hasil yang lebih terprediksi. Dalam *Snake*, pemain yang terus-menerus menggunakan strategi agresif untuk langsung mendapatkan buah mungkin lebih cepat gagal, sementara pemain yang lebih berhati-hati dapat bertahan lebih lama dengan strategi yang lebih defensif, seperti menjaga ruang gerak yang luas sebelum mengincar buah (Myerson, 1991).

2.5 Deep Learning

Deep Learning (DL) adalah teknik yang menggunakan jaringan saraf tiruan berlapis-lapis (deep neural networks) untuk menangani masalah non-linear yang kompleks. Pada kasus gim seperti *Snake*, RL digunakan agar agen (dalam hal ini, pemain *Snake*) belajar bagaimana bergerak dalam grid untuk memaksimalkan *reward*—biasanya dengan makan makanan dan menghindari benturan dengan tubuh atau dinding. Teori-teori utama yang ada pada Deep *Reinforcement Learning* (DRL) diantaranya adalah Deep-Q-Network (*DQN*), Experience Replay, Target Network, Policy Gradient Method.

2. 5. 1. Deep-Q-Network (*DQN*)

Banyak aplikasi dari *Reinforcement Learning* seperti mobil yang dapat menyetir sendiri, Atari, konsumen marketing, pelayanan kesehatan, edukasi dan lainnya memiliki banyak *state* (s) dan atau aksi (a). Bahkan ada bisa saja ada beberapa state yang tidak dijumpai saat model training dijalankan. Hal ini membutuhkan model representatif seperti *reward*, nilai dari *state-action* (Q), nilai dari state (V), dan peraturan yang dapat memunculkan beberapa state dan aksi. Agen mempelajari VFA (*Value*

Function Approximation) dengan memperbarui bobot setiap langkah waktu berdasarkan *Temporal Difference* (TD) return saat ini. Setelah pembaruan, TD return tersebut dibuang, dan TD return baru digunakan untuk pembaruan bobot berikutnya. Metode ini tidak memaksimalkan penggunaan data yang telah diambil untuk menyesuaikan fungsi.

Istilah "Deep *Q-Learning*" digunakan karena jaringan saraf biasanya diimplementasikan dengan deep neural networks (DNNs) dalam banyak aplikasi. TD (*Temporal Difference*) adalah metode dalam *Reinforcement Learning* yang digunakan untuk memperkirakan nilai (*value*) dari suatu keadaan atau aksi berdasarkan perbedaan temporal, yaitu perbedaan antara nilai yang diprediksi saat ini dan nilai yang sebenarnya diterima di langkah berikutnya. TD menggabungkan aspek dari *Monte Carlo* dan *dynamic programming*.

Inti dari metode TD adalah TD error, yang mengukur kesalahan prediksi antara nilai saat ini dan estimasi nilai berikutnya. Pembaruan dilakukan secara bertahap setiap kali agen mengambil tindakan dan menerima umpan balik (*reward*). *TD learning* sering digunakan dalam algoritma seperti *Q-Learning* dan SARSA.

Berikut adalah contoh sederhana dari Update Temporal Difference:

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)] \quad \text{.....(2.2)}$$

Persamaan 2.2 merupakan rumus pembaruan nilai keadaan dalam pembelajaran penguatan (reinforcement learning). Dalam persamaan ini, V(s) merepresentasikan nilai dari suatu keadaan s, sedangkan α adalah laju pembelajaran yang menentukan seberapa besar perubahan nilai berdasarkan pengalaman baru. Nilai r merupakan *reward* atau imbalan yang diterima saat berpindah dari keadaan s ke keadaan s′. Selanjutnya, γ / *gamma* adalah faktor diskon yang menunjukkan seberapa besar pengaruh nilai masa depan terhadap nilai saat ini. Terakhir, V(s′) adalah nilai prediksi untuk keadaan berikutnya s′.

Dengan menggunakan pendekatan ini, agen belajar dari pengalaman tanpa perlu menunggu episode selesai, berbeda dengan metode *Monte Carlo* yang memerlukan penyelesaian episode penuh untuk memperbarui nilai (Kuang, 2021).

2. 5. 2. Experience Replay

Pada metode Experience Replay dalam Reinforcement Learning, agen menyimpan data pengalaman interaksi dengan lingkungan dalam bentuk tuple (s, a, r, s′), yang masing-masing merepresentasikan keadaan saat ini (*state*), aksi yang diambil (*action*), reward yang diterima (*reward*), dan keadaan berikutnya (*next state*). Data ini disimpan ke dalam replay buffer, yaitu struktur data khusus yang berfungsi sebagai memori jangka pendek agen. Dengan menyimpan dan mendaur ulang pengalaman ini, algoritma dapat melakukan pembaruan nilai Q secara lebih stabil dan efisien. Pengambilan sampel secara acak dari *buffer* ini bertujuan untuk mengurangi korelasi antar data dan memungkinkan pemanfaatan ulang informasi berharga dari interaksi sebelumnya. Hal ini menjadikan *Experience Replay* sebagai teknik penting untuk meningkatkan efisiensi pembelajaran, terutama dalam lingkungan yang sangat dinamis seperti permainan Snake.

Pengambilan sampel dari sebagian kecil tuple untuk melatih jaringan saraf disebut *experience replay*. Selain mengurangi korelasi, *experience replay* memungkinkan kita belajar lebih banyak dari satu set tuple berkali-kali, dan secara umum memaksimalkan penggunaan pengalaman.

Untuk melakukan *experience replay*, kita secara acak mengambil sampel *batch* dari tuple pengalaman, menghitung nilai target, dan kemudian menggunakan *Stochastic Gradient Descent* (SGD) untuk memperbarui bobot jaringan saraf. Metode pelatihan jaringan saraf umum dapat digunakan untuk tujuan ini (Kuang, 2021).

2. 5. 3. Policy Gradient Method

Metode *policy gradient* mempelajari kebijakan secara langsung, sementara metode seperti TD, SARSA, dan *Q-Learning* adalah metode berbasis nilai, yang berfokus pada evaluasi dan pembaruan nilai dari tindakan atau keadaan tertentu. Teorema *policy gradient* adalah konsep penting dalam metode *policy gradient*, yang akan dijelaskan dengan bukti terperinci. Sebelum itu, dijelaskan contoh sederhana berupa MDP satu langkah, di mana proses dimulai dari keadaan tertentu dan selesai setelah satu langkah dengan pemberian *reward* $r = R_{s,a}$.

Fungsi objektifnya adalah:

$$J(\theta) = \mathrm{E}_{\pi\theta}[r] = \sum_{s\in S} d(s) \sum_{a\in A} \pi_\theta(s,a) R_{s,a} \quad \text{.....(2.3)}$$

Persamaan 2.3 ini menggambarkan cara menghitung ekspektasi *reward* total yang diharapkan (dinyatakan sebagai $J(\theta)$) dengan menjalankan kebijakan tertentu π_θ. Ekspektasi ini dihitung dengan menjumlahkan *reward* dari setiap pasangan keadaan-tindakan (s, a), dikalikan probabilitas kebijakan untuk memilih tindakan tertentu pada suatu keadaan ($\pi_\theta(s, a)$), dan frekuensi terjadinya $d(s)$. Teorema *Policy gradient* pada dasarnya mencoba untuk mengoptimalkan kebijakan dengan mengubah parameter θ untuk memaksimalkan *reward* $J(\theta)$ yang akan dihasilkan (Kuang, 2021).

2.6 Penelitian Terdahulu

Berdasarkan literatur yang digunakan dalam penulisan proposal ini, dapat disimpulkan bahwa implementasi reinforcement learning dapat membantu sistem dalam mengambil keputusan secara adaptif berdasarkan pengalaman, sehingga meningkatkan efektivitas dalam proses pengambilan keputusan. Tabel 2.1 ini adalah acuan yang memiliki keterkaitan dengan penelitian yang akan dilakukan.

Tabel 2.1 Penelitian Terdahulu

No	Nama Penulis dan Tahun Publikasi	Hasil
1	(Bakar, dkk., 2023)	**Judul**: *Fusion Sparse And Shaping Reward Function In Soft Actor-Critic Deep Reinforcement Learning For Mobile Robot Navigation* **Metode**: *Soft Actor Critic* **Hasil:** Saat ini, kemajuan dalam robot otonom sangat dipengaruhi oleh perkembangan teknologi terbaru. *Deep Reinforcement Learning* (DRL) memungkinkan sistem beroperasi secara otomatis, di mana robot mempelajari gerakan berdasarkan interaksi dengan lingkungan. Salah satu pendekatan DRL terbaru adalah *Soft Actor Critic* (SAC) yang mampu mengontrol tindakan berkelanjutan untuk menghasilkan gerakan yang lebih akurat. Meskipun SAC tangguh terhadap ketidakpastian, terdapat kelemahan dalam proses eksplorasi yang mempengaruhi kecepatan pembelajaran. Penelitian ini mengusulkan penggunaan fungsi ganjaran berbasis ganjaran jarang dan terstruktur dalam metode SAC untuk meningkatkan efektivitas pembelajaran robot. Hasil eksperimen menunjukkan bahwa kombinasi ini berhasil menavigasi robot ke sasaran dengan meningkatkan akurasi hingga rata-rata kesalahan 4,99%.
2	(Ridho dkk., 2024)	**Judul**: Penerapan *Deep Reinforcement Learning* dalam *Business Intelligence* **Metode:** *Deep Neural Network*

No	Nama Penulis dan Tahun Publikasi	Hasil
		Hasil: *Business Intelligence* (BI) adalah kombinasi alat seperti gudang data, OLAP, dan dasbor untuk analisis data mendalam. Gudang data mengumpulkan data akurat dari berbagai sumber, sementara OLAP memungkinkan analisis multidimensi secara real-time. *Deep Reinforcement Learning* (DRL) diidentifikasi sebagai teknik yang menjanjikan untuk mengatasi tantangan pengambilan keputusan berurutan dengan ketidakpastian. *Deep neural network* (DNN) juga digunakan dalam BI untuk prediksi yang lebih baik. Temuan menunjukkan bahwa *Double Deep Q Learning* dapat meningkatkan BI dengan mengoptimalkan sumber daya dan mengurangi waktu kalkulasi dalam masalah skala besar serta memodelkan pola strategi lawan dalam sistem multi-agen.
3	(Afriyadi & Utomo, 2023)	**Judul**: *Towards Human-Level Safe Reinforcement Learning in Atari Library* **Metode**: *Double Deep Q Learning* **Hasil**: Penelitian ini mengusulkan pendekatan baru untuk mengintegrasikan pembatasan keselamatan yang dibuat secara manual dalam algoritme *Double Deep Q-Network* (D*DQN*). Tujuannya adalah agar agen RL dapat belajar untuk bertindak dengan aman tanpa mengorbankan kinerja. Penelitian ini menggunakan pustaka Atari, khususnya gim *Super Mario Bros*, sebagai contoh lingkungan yang kompleks untuk melatih agen RL. Hasil

No	Nama Penulis dan Tahun Publikasi	Hasil
		penelitian menunjukkan bahwa metode yang diusulkan, termasuk pembatasan keselamatan, mampu mengungguli algoritme RL konvensional dalam hal keamanan dan pencapaian hasil yang optimal.
4	(Taqwa dkk., 2023)	**Judul**: *Designing A WSNs-based Smart Home Monitoring System through Deep Reinforcement Learning* **Metode**: *Deep Reinforcement Learning* **Hasil**: pengembangan sistem pemantauan rumah pintar berbasis *wireless sensor networ*ks (WSNs) dan *deep Reinforcement Learning* (DRL) untuk memantau parameter lingkungan seperti suhu, kelembapan, dan kadar CO2. Sistem ini dirancang untuk memberikan rekomendasi apakah lingkungan rumah dalam kondisi "nyaman" atau "tidak nyaman." Proses eksperimen dilakukan melalui validasi akurasi sensor WSNs, pemilihan model DRL terbaik, dan pengujian sistem pemantauan. Hasil pengujian menunjukkan bahwa sistem ini memiliki tingkat akurasi tinggi (95.52% untuk DRL, 98.52% untuk WSNs) dan biaya rendah, serta mudah diimplementasikan. Sistem ini juga dilengkapi aplikasi berbasis Android untuk memudahkan pengguna dalam memantau kondisi rumah secara *real-time*.
5	(Hidayat dkk., 2023)	**Judul**: *Modified Q-Learning Algorithm for Mobile Robot Real-Time Path Planning using Reduced States* **Metode**: *Q-Learning*

No	Nama Penulis dan Tahun Publikasi	Hasil
		Hasil: Perencanaan jalur adalah algoritme penting dalam robot otonom, termasuk robot pertanian. Salah satu metode *Reinforcement Learning* yang digunakan untuk perencanaan jalur robot adalah algoritme *Q-Learning*. Namun, metode *Q-Learning* konvensional membutuhkan biaya komputasi tinggi karena mengeksplorasi semua kemungkinan keadaan robot untuk menemukan jalur optimal. Studi ini memodifikasi algoritme *Q-Learning* dengan menghilangkan area yang tidak dapat dilalui sehingga area tersebut tidak dihitung sebagai grid yang harus diproses. Algoritme *Q-Learning* yang dimodifikasi ini diuji pada robot otonom di Agribusiness and Technology Park (ATP), IPB. Hasil simulasi menunjukkan bahwa modifikasi ini mengurangi biaya komputasi sebesar 50,71%, dengan waktu rata-rata 25,74 detik dibandingkan dengan 50,75 detik pada metode *Q-Learning* asli. Meskipun begitu, kedua metode menghasilkan jumlah keadaan yang sama untuk jalur optimal, yaitu 56 keadaan. Algoritme *Q-Learning* yang dimodifikasi juga dapat menemukan jalur dengan nilai parameter learning rate minimal 0,2 ketika faktor diskon bernilai 0,9.
6	(Zhu & Zhang, 2021)	**Judul**: *Deep Reinforcement Learning Based Mobile Robot Navigation: A Review* **Metode**: *Deep Q Network* **Hasil**: Aplikasi pelaporan dan pelacakan kejahatan berhasil dibuat. Masyarakat dapat melaporkan pelaku kejahatan secara *realtime*

No	Nama Penulis dan Tahun Publikasi	Hasil
		melalui *smartphone* android. Kejahatan akan cepat ditangani menggunakan konsep *geofencing.* Di dalam aplikasi ini memiliki 2 jenis *user* yaitu masyarakat dan polisi, dengan adanya fitur *near me* masyarakat dan polisi dapat melihat apa yang terjadi di sekitarnya. Masyarakat harus *login* terlebih dahulu untuk menggunakan fitur *panic button*.
7	(Alrahma dkk., 2024)	**Judul**: *Application of Q-Learning Method for Disaster Evacuation Route Design Case Study: Digital Center Building UNNES* **Metode**: *Q-Learning* **Hasil**: Penelitian ini menerapkan *Q-Learning*, sebuah teknik pembelajaran penguatan, untuk merancang rute evakuasi secara efisien di lantai pertama Gedung Digital Center di UNNES, yang menghasilkan jalur optimal dengan akurasi tinggi dan efisiensi komputasi yang baik, bahkan ketika ada hambatan tambahan.
8	(Changbao dkk., 2024)	**Judul**: *Research on Game Character Behavior Decision-Making System Based on Deep Reinforcement Learning* **Metode**: *Deep Reinforcement Learning* **Hasil**: Makalah ini mengeksplorasi penggunaan *Deep Reinforcement Learning* (DRL) dalam meningkatkan kemampuan pengambilan keputusan karakter gim. Metode tradisional seperti Pohon Perilaku dan Mesin Keadaan Terbatas memiliki keterbatasan dalam adaptabilitas dan kinerja karena sifatnya yang

No	Nama Penulis dan Tahun Publikasi	Hasil
		berbasis aturan. Studi ini menyajikan model DRL, khususnya menggunakan *Deep Q-Network* (*DQN*), yang menunjukkan peningkatan kualitas pengambilan keputusan dan kecepatan respons untuk karakter gim. Penelitian ini membahas penerapan DRL dalam lingkungan gim yang kompleks, menekankan keunggulannya seperti strategi perilaku adaptif dan pengalaman bermain gim yang lebih baik. Meskipun memiliki potensi, DRL menghadapi tantangan seperti waktu pelatihan yang panjang dan interpretabilitas model yang buruk

BAB 3
METODOLOGI PENELITIAN

Pada bab ini dijelaskan metodologi yang digunakan dalam penelitian ini. Metodologi mencakup pendekatan dan langkah-langkah yang sistematis untuk mencapai tujuan penelitian, khususnya dalam konteks penerapan *Reinforcement Learning*.

Pengerjaan penelitian ini meliputi garis besar penelitian, diagram alir proses, prosedur penelitian, serta rencana jadwal pelaksanaan. Seluruh tahapan tersebut akan digunakan dalam **"IMPLEMENTASI *ARTIFICIAL INTELLIGENCE* PADA PERMAINAN SNAKE DENGAN METODE *REINFORCEMENT LEARNING*"**.

3.1 Gambaran Besar Penelitian

Gambaran besar penelitian ini adalah memberikan panduan umum mengenai alur penelitian dari awal hingga akhir. Pengerjaan "Implementasi *Artificial intelligence* pada Permainan *Snake* dengan Metode *Reinforcement Learning*" dibagi menjadi 4 tahap.

Tahap pertama adalah pengembangan lingkungan simulasi permainan *Snake* yang akan digunakan sebagai platform pelatihan untuk agen AI. Pada tahap ini, dilakukan pemodelan permainan *Snake* dengan mempertimbangkan semua variabel lingkungan, seperti ukuran papan permainan, posisi makanan, dan kondisi-kondisi permainan yang memungkinkan. Simulasi ini dirancang agar agen dapat berinteraksi dengan lingkungan secara efektif untuk mempelajari tindakan yang menghasilkan *reward* optimal.

Tahap kedua adalah penerapan algoritma *Reinforcement Learning*, seperti *Q-Learning* dan *Deep Q-Network* (*DQN*), pada lingkungan simulasi tersebut. Agen AI dilatih dengan berbagai iterasi permainan, di mana setiap tindakan yang diambil menghasilkan *reward* atau penalti. Pada tahap ini, algoritma *Q-Learning* akan menggunakan tabel nilai (Q-table) untuk menyimpan *reward* potensial dari setiap

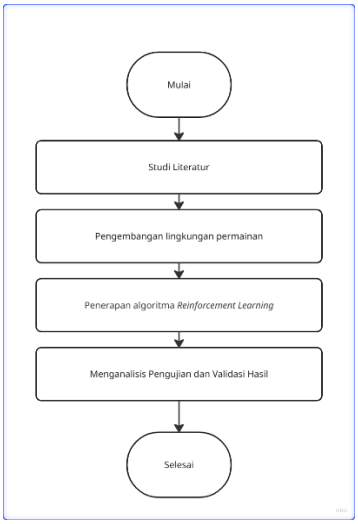
tindakan, sementara metode *DQN* akan memanfaatkan jaringan saraf tiruan untuk mengestimasi nilai Q dalam ruang tindakan yang lebih kompleks.

Tahap ketiga adalah analisis kinerja agen AI yang dilatih menggunakan algoritma-algoritma tersebut. Setelah agen selesai dilatih, dilakukan evaluasi terhadap kemampuan agen dalam memainkan permainan *Snake*, meliputi seberapa baik agen dapat menghindari tabrakan, memakan makanan, dan memaksimalkan skor. Hasil evaluasi ini akan menunjukkan efektivitas algoritma *Reinforcement Learning* yang digunakan.

Tahap keempat adalah pengujian dan validasi hasil. Pada tahap ini, dilakukan analisis konsistensi kinerja agen dan perbandingan antara hasil pelatihan menggunakan *Q-Learning* dan *DQN*. Diperiksa apakah hasil yang diperoleh konsisten dalam berbagai kondisi lingkungan permainan yang berbeda dan apakah agen menunjukkan kemampuan adaptasi yang baik terhadap perubahan. Hasil dari tahap ini akan menjadi dasar untuk menarik kesimpulan mengenai efektivitas metode *Reinforcement Learning* dalam permainan *Snake* dan potensi pengembangannya untuk aplikasi lain.

3.2 Diagram Alir Penelitian

Adapun diagram alir dari penelitian ini adalah sebagai berikut:



Gambar 3. 1 Diagram Alir Penelitian

Pada Gambar 3.1 ditunjukkan diagram alir dari penelitian ini. Terdapat 4 tahapan, yaitu Pengembangan lingkungan simulasi, Algoritma *Reinforcement Learning*, Analisis kinerja agen AI, dan pengujian dan validasi hasil.

3.3 Prosedur Penelitian

Adapun penjelasan lebih lengkap terkait tahapan-tahapan yang ada pada sub bab diagram alir penelitian (Gambar 3.1) akan dijelaskan pada sub bab prosedur penelitian ini. Berikut penjelasan tahap-tahap yang dilakukan untuk menyelesaikan penelitian implementasi *artificial intelligence* pada permainan *snake* dengan metode *Reinforcement Learning*.

3.3.1 Studi Literatur

Studi literatur ini terdapat pada tahap pertama untuk mengumpulkan literasi yang sesuai dengan topik penelitian dalam implementasi *artificial intelligence* pada permainan *snake* dengan metode *Reinforcement Learning*. Beberapa literatur yang akan digunakan adalah tentang metode Double Q Network, penjelasan mengenai agen AI.

Adapun hasil dari tahap studi literatur adalah didapatkan penjelasan terkait metode, dan didapatkan informasi terkait dengan metode yang tepat serta kelebihan dari metode tersebut dalam proses pengembangan sistem model AI dengan *Reinforcement Learning*. Kemudian memberikan gambaran terkait tahapan metodologi yang harus dilalui berdasarkan metode yang digunakan.

3.3.2 Pengembangan Lingkungan Simulasi

Dalam tahap pengembangan lingkungan simulasi, kami akan membuat model permainan *Snake* yang komprehensif dengan mempertimbangkan berbagai variabel penting seperti ukuran papan permainan, posisi makanan, dan kondisi permainan yang beragam. Simulasi ini dirancang untuk memungkinkan agen kecerdasan buatan (AI) berinteraksi secara efektif dengan lingkungan, sehingga dapat mempelajari tindakan yang menghasilkan *reward* optimal. Dengan

lingkungan simulasi yang realistis dan dinamis, kami bertujuan untuk menciptakan platform pelatihan yang ideal bagi agen AI untuk mengembangkan strategi bermain yang efisien dan adaptif.

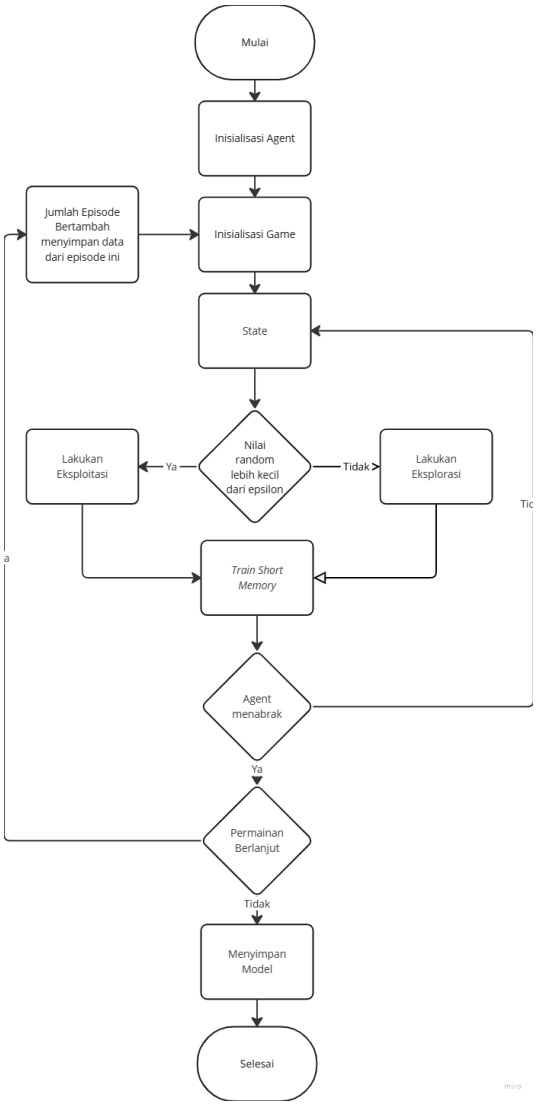
Agar simulasi ini dapat mendukung proses pembelajaran agen AI dengan maksimal, kami akan memastikan setiap elemen permainan *Snake* disimulasikan secara akurat. Ini termasuk penyesuaian tingkat kesulitan permainan dan penyediaan umpan balik yang tepat untuk setiap tindakan yang dilakukan oleh agen. Selain itu, simulasi ini akan dioptimalkan untuk memungkinkan iterasi yang cepat, memungkinkan agen untuk belajar dari pengalaman dengan cepat dan efektif.

3.3.3 Penerapan Algoritma *Reinforcement Learning*

Pada tahap penerapan algoritma dalam penelitian "Implementasi *Artificial intelligence* pada Permainan *Snake* dengan Metode *Reinforcement Learning*", beberapa langkah penting dilakukan untuk memastikan agen AI dapat belajar dan beradaptasi dengan lingkungan permainan secara efektif.

Algoritma *Reinforcement Learning* yang dipilih untuk diterapkan adalah *Q-Learning* dan *Deep Q-Network* (*DQN*). *Q-Learning* menggunakan tabel nilai (Q-table) untuk menyimpan *reward* potensial dari setiap tindakan, sedangkan *DQN* memanfaatkan jaringan saraf tiruan untuk mengestimasi nilai Q dalam ruang tindakan yang lebih kompleks. Algoritma tersebut diterapkan pada lingkungan simulasi yang telah dikembangkan. Proses ini melibatkan pengkodean algoritma dan integrasinya dengan simulasi permainan *Snake*, sehingga agen dapat mulai belajar dari interaksi dengan lingkungan. Agen AI dilatih melalui berbagai iterasi permainan. Setiap tindakan yang diambil oleh agen menghasilkan *reward* atau penalti, yang digunakan untuk memperbarui strategi agen. Proses ini memungkinkan agen untuk belajar dari pengalaman dan meningkatkan kemampuannya dalam bermain *Snake*. Selama pelatihan, dilakukan pengujian untuk memantau kinerja agen dan menyesuaikan parameter algoritma jika diperlukan. Hal ini penting untuk memastikan bahwa agen dapat mencapai performa optimal dalam permainan.

Skema yang akan digunakan adalah sebagai berikut:



Gambar 3. 2 Skema *Deep Q Network*

Gambar 3.2 menunjukkan skema alur proses pembelajaran agen pada permainan Snake dengan menggunakan metode *Reinforcement Learning* berbasis *Deep Q-Network* (*DQN*). Proses diawali dengan inisialisasi agen yang mencakup pengaturan parameter seperti jumlah episode pelatihan, nilai epsilon sebagai tingkat eksplorasi, *learning rate*, *discount rate*. Setelah agen siap, dilakukan inisialisasi terhadap lingkungan permainan Snake, yang meliputi penempatan posisi awal ular, makanan, serta arah gerak awal.

Setelah permainan dimulai, agen akan mengamati kondisi lingkungan saat ini (state) yang berisi informasi seperti arah gerak ular, posisi makanan relatif terhadap kepala ular, serta potensi tabrakan dengan dinding atau tubuhnya sendiri. Berdasarkan kondisi tersebut, agen akan menentukan aksi selanjutnya dengan pendekatan epsilon-greedy. Jika nilai acak lebih kecil dari epsilon, maka agen memilih aksi secara acak (eksplorasi), sedangkan jika lebih besar, agen memilih aksi berdasarkan prediksi terbaik dari model (eksploitasi).

Aksi yang diambil akan dijalankan dalam permainan dan menghasilkan reward serta perubahan lingkungan. Pengalaman yang terdiri dari state, action, reward, next state, dan kondisi selesai atau tidak akan langsung digunakan untuk pelatihan memori jangka pendek (*short-term memory*), dan juga disimpan dalam replay buffer untuk keperluan pelatihan jangka panjang (*long-term memory*). Ketika permainan belum selesai, agen akan kembali mengamati state baru dan melanjutkan proses pengambilan keputusan. Namun jika permainan berakhir, misalnya karena agen menabrak dinding atau tubuhnya sendiri, maka sistem akan melakukan pelatihan memori jangka panjang menggunakan batch sampel dari *replay buffer*.

Setiap akhir episode, agen juga melakukan evaluasi performa dan menyimpan model terbaik jika terjadi peningkatan *score*. Proses ini diulang hingga jumlah episode yang ditentukan terpenuhi. Setelah pelatihan selesai, model disimpan untuk digunakan kembali pada tahap pengujian atau implementasi lanjutan. Skema ini dirancang untuk memastikan bahwa agen mampu belajar secara bertahap dari interaksi dengan lingkungan, dengan tujuan akhir yaitu menghasilkan agen yang `, optimal, dan mampu bertahan hidup dalam permainan sembari memaksimalkan *reward*.

3.3.4 Menganalisis Pengujian dan Validasi Hasil

Setelah model agen AI dilatih menggunakan algoritma Deep Q-Network (DQN) dalam permainan Snake, tahap berikutnya adalah melakukan pengujian guna mengevaluasi performa agen yang telah dilatih. Pengujian ini dilakukan dengan membiarkan agen bermain dalam lingkungan permainan yang sama, tanpa adanya modifikasi terhadap

parameter atau aturan permainan. Tujuannya adalah untuk menilai seberapa efektif agen dalam mengambil keputusan dan bertahan hidup berdasarkan pembelajaran yang telah diperoleh selama pelatihan.

Selama proses pengujian, beberapa metrik penting digunakan sebagai indikator keberhasilan, di antaranya adalah jumlah skor yang diperoleh, lama waktu agen bertahan hidup (jumlah langkah), serta konsistensi dalam pengambilan keputusan. Pengujian ini juga memberikan gambaran apakah model mampu menggeneralisasi strategi yang efektif terhadap berbagai situasi permainan atau hanya sekadar mengingat pola yang sering muncul selama pelatihan.

Selain pengujian, dilakukan pula validasi terhadap hasil pelatihan untuk memastikan bahwa model tidak mengalami overfitting dan bahwa strategi yang dikembangkan oleh agen AI memang berasal dari pembelajaran yang bermakna, bukan kebetulan atau eksploitasi dari kondisi permainan yang sempit. Validasi dilakukan dengan mengulangi permainan beberapa kali menggunakan seed acak yang berbeda, serta membandingkan performa agen dengan baseline atau model acak.

Hasil pengujian dan validasi menunjukkan bahwa agen yang dilatih dengan DQN menunjukkan peningkatan skor rata-rata dan kemampuan bertahan hidup yang lebih tinggi dibandingkan dengan agen acak. Hal ini menunjukkan bahwa penggunaan algoritma Reinforcement Learning dalam permainan Snake memberikan kontribusi signifikan dalam membentuk strategi yang adaptif dan efisien. Dari sini dapat disimpulkan bahwa agen AI telah berhasil belajar dari lingkungannya dan dapat mengambil keputusan yang mengarah pada perolehan reward yang lebih besar.

3.4 Jadwal Penelitian

Adapun susunan jadwal penelitian yang dilakukan oleh penulis berdasarkan metodologi yang telah dijabarkan sebelumnya. Susunan penelitian dapat dilihat pada Tabel 3.1 sebagai berikut.

Tabel 3.1 Jadwal Penelitian

Tahap	Bulan											
	September				Oktober				November			
	1	2	3	4	1	2	3	4	1	2	3	4
Melakukan studi literasi												
Melakukan simulasi pengembangan lingkungan agen												
Implementasi algoritma												
Melakukan pelatihan pada agen												
Analisis performa agen												
Melakukan testing dan validasi												
Hasil Analisa pembelajaran agen												
Penulisan laporan												

BAB 4
HASIL DAN PEMBAHASAN

Pada bab 4 ini akan dibahas secara menyeluruh hasil yang diperoleh dari implementasi metode *Reinforcement Learning*, khususnya *Deep Q-Network* (DQN) dalam permainan Snake, serta analisis atas performa agen yang telah dilatih. Penelitian ini mencakup proses pelatihan agen menggunakan lingkungan simulasi yang dikembangkan dengan PyGame, pengaturan parameter pelatihan seperti learning rate, discount factor, dan exploration rate, serta perbandingan antara dua pendekatan pembelajaran, yaitu Bellman dan Monte Carlo. Setiap hasil yang diperoleh selama pelatihan didokumentasikan dalam bentuk grafik dan data metrik performa untuk mengevaluasi efektivitas dan efisiensi algoritma yang digunakan dalam menghasilkan agen yang mampu memainkan permainan dengan baik.

Selain memaparkan hasil kuantitatif, bab ini juga menginterpretasikan perilaku agen berdasarkan hasil pengamatan visual yang dilakukan selama simulasi. Perilaku agen dalam merespons kondisi lingkungan, strategi yang berkembang seiring proses pelatihan, serta adaptasi terhadap tantangan permainan menjadi fokus utama dalam pembahasan ini. Evaluasi menyeluruh terhadap keberhasilan maupun keterbatasan dari metode yang diterapkan akan diuraikan untuk memberikan pemahaman yang lebih mendalam mengenai implementasi *Reinforcement Learning* dalam konteks permainan Snake. Hasil ini juga menjadi dasar dalam menarik kesimpulan dan rekomendasi untuk pengembangan selanjutnya pada bab akhir.

4.1. Hasil Implementasi

Implementasi algoritma *Reinforcement Learning* dalam permainan Snake dilakukan dengan pendekatan *Deep Q-Network (DQN),* di mana dua model utama diterapkan untuk membandingkan efektivitasnya, yaitu model berbasis pendekatan *Bellman* dan pendekatan *Monte Carlo*. Lingkungan simulasi dikembangkan menggunakan PyGame, dan dirancang sedemikian

rupa agar agen dapat belajar secara otonom melalui interaksi langsung dengan lingkungan, memperoleh reward saat berhasil memakan makanan, dan mendapatkan penalti saat menabrak dinding atau tubuhnya sendiri.

Proses pelatihan dilakukan sebanyak 5000 episode untuk masing-masing pendekatan. Pada model *Bellman*, seperti terlihat pada Gambar 4.1, agen menunjukkan peningkatan performa yang cukup tajam pada 1000 episode pertama. Setelah itu, grafik reward mulai stagnan dengan rata-rata skor berkisar antara 23 hingga 24. Hal ini menunjukkan bahwa agen telah menemukan strategi dasar yang efektif, namun belum cukup optimal untuk mencapai peningkatan performa yang signifikan. Fluktuasi pada reward per episode masih cukup tinggi, menandakan bahwa agen belum sepenuhnya stabil dalam setiap permainan.

Sementara itu, model Monte Carlo, seperti diperlihatkan pada Gambar 4.3, menunjukkan performa awal yang kurang stabil pada 500 episode pertama. Namun, setelah melewati fase awal tersebut, performa meningkat dan mulai menunjukkan konsistensi pada episode ke-1000 hingga ke-1500. Setelah fase itu, rata-rata reward juga menunjukkan pola stagnan, serupa dengan model *Bellman*, meskipun proses peningkatan *reward* berlangsung lebih lambat. Ini menunjukkan bahwa meskipun *Monte Carlo* membutuhkan waktu lebih lama untuk belajar, tetapi tetap mampu mencapai hasil yang setara dalam jangka panjang.

Secara kuantitatif, hasil pelatihan menunjukkan bahwa agen yang dilatih menggunakan pendekatan Bellman mencapai skor rata-rata sebesar 23,58 dan panjang maksimum ular mencapai 70. Hasil ini menunjukkan bahwa agen berhasil mempelajari strategi bertahan hidup dan perolehan *reward*, namun belum cukup efisien dalam menemukan strategi eksplorasi yang lebih agresif. Visualisasi perilaku agen juga menunjukkan pola pergerakan yang semakin terarah ke makanan, dengan kecenderungan membentuk pola defensif seperti bergerak mengikuti tepi arena permainan.

Dengan hasil ini, dapat disimpulkan bahwa algoritma DQN mampu menghasilkan agen yang mampu bermain permainan Snake secara mandiri

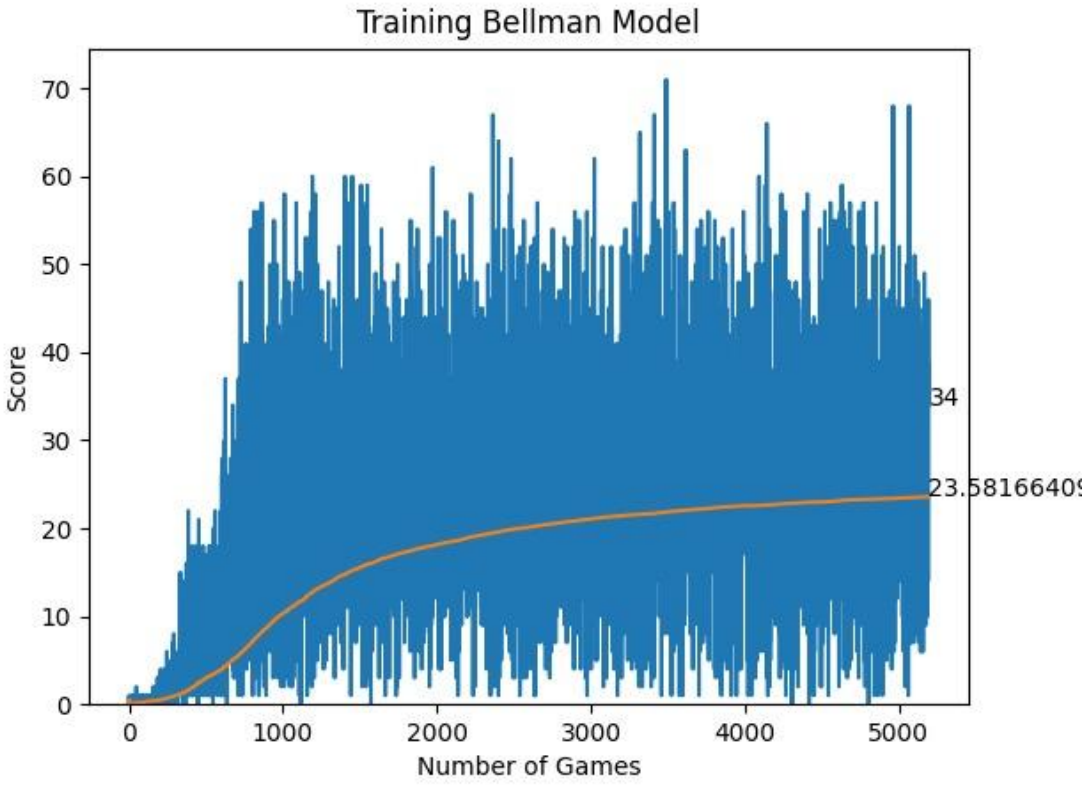
dengan performa yang cukup stabil. Namun, stagnasi performa menunjukkan bahwa diperlukan tuning parameter lebih lanjut atau pendekatan algoritmik yang lebih kompleks untuk mencapai hasil yang lebih optimal.

4.1.1. Hasil Pelatihan Model

Proses pelatihan agent pada permainan Snake dilakukan dengan menggunakan metode *Reinforcement Learning* berbasis Deep Q-Network (DQN). Model dilatih dalam lingkungan permainan Snake yang dibangun dengan Pygame. Model yang digunakan merupakan model Bellman dan Monte Carlo.

Pelatihan dilakukan selama lebih dari 5000 repetisi dengan setiap episode berakhir ketika agent gagal. Selama proses pelatihan, nilai reward dan skor rata-rata dicatat untuk memantau perkembangan agen. Berikut ini adalah beberapa hasil penting dari proses pelatihan bellman:

- Model dilatih dengan data dari sekitar 5000 episode.
- Skor model meningkat pesat di awal, lalu cenderung stabil atau stagnan di sekitar rata-rata 23–24.
- Masih terdapat banyak fluktuasi dalam performa per game, meskipun rata-ratanya meningkat.
- Model sudah belajar cukup baik, tapi mungkin belum optimal.
- Potensi ada ruang untuk peningkatan dengan eksplorasi yang lebih baik atau parameter pelatihan yang dituning lebih lanjut.
- Grafik per episode dapat dilihat pada Gambar 4.1, yang memperlihatkan tren peningkatan performa seiring bertambahnya episode pelatihan yang dibuktikan dengan nilai rata-rata yang selalu naik.



Gambar 4. 1 Grafik Training Model *Bellman* dengan 5000 episode

Dari gambar 4.1 kita dapat melihat tren nilai rata-rata yang terus meningkat. Kita dapat melihat peningkatan signifikan pada 1000 episode pertama. Model menjadi underfitting karena nilai rata-rata stagnan di sekitar 23-24. Nilai dari setiap episode juga sangat berfluktuasi

4.1.2. Performa Agen dalam Permainan

Setelah proses pelatihan selesai, agent diuji sebanyak lebih dari 5000 kali untuk menilai konsistensi performa. Dari pengujian tersebut, diperoleh hasil sebagai berikut:

- Skor rata-rata: 23,58
- Panjang ular maksimum: 70

Performa ini menunjukkan bahwa agent mampu mengembangkan strategi dasar yang cukup efektif, seperti menghindari dinding, tidak menabrak

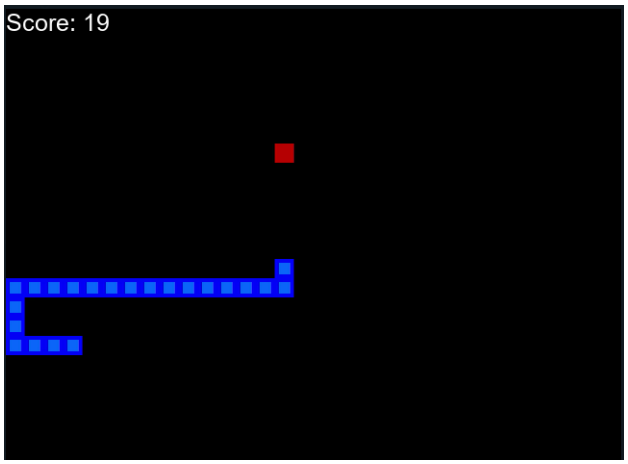
tubuh sendiri, dan secara efisien mendekati makanan. Walaupun agent mendapat kenaikan yang stagnan setelah pengujian ke-1000.

4.1.3. Visualisasi Perilaku Agen

Untuk memberikan gambaran visual terhadap perilaku agent, dilakukan dokumentasi dalam bentuk tangkapan layar dan/atau animasi selama agent bermain. Beberapa perilaku yang diamati secara konsisten antara lain:

- Agent belajar untuk menjaga jarak aman dari dinding dan ekornya.
- Gerakan agent semakin halus dan terarah menuju makanan seiring waktu pelatihan.
- Agent terkadang membuat pola gerak defensif seperti berputar atau mengikuti tepi permainan.

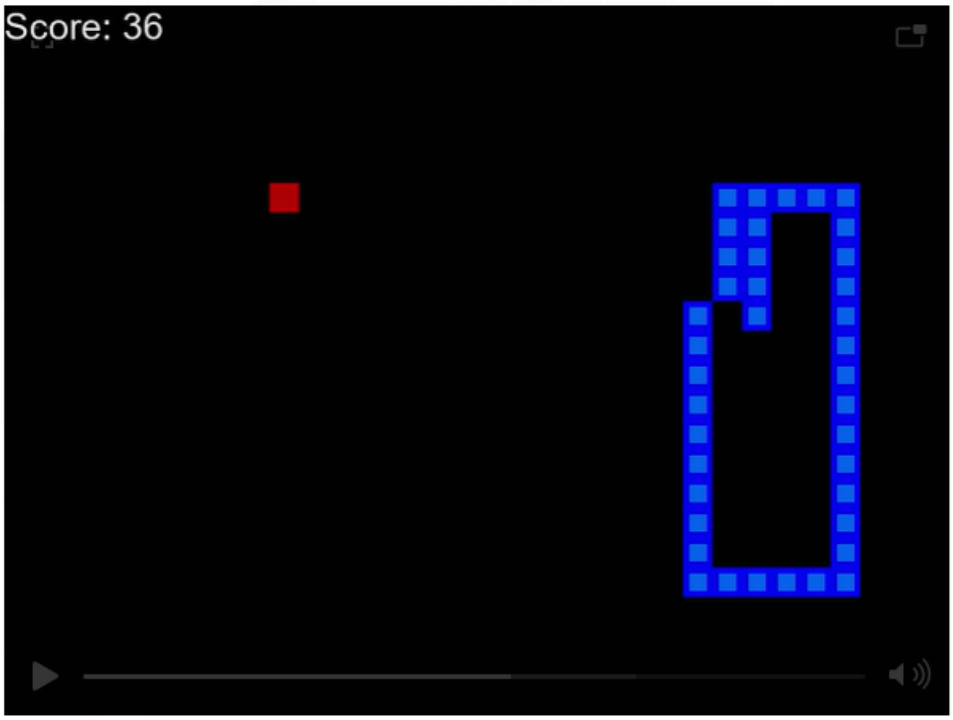
Contoh visualisasi dapat dilihat pada Gambar 4.2 dan 4.3 berikut:



Gambar 4. 2 Agent sedang mendekati makanan secara strategis

Gambar ini memperlihatkan salah satu momen penting dalam proses pengujian agen setelah pelatihan, di mana agen berhasil menunjukkan perilaku adaptif dengan mendekati posisi makanan secara strategis. Dalam situasi tersebut, agen telah menghindari potensi tabrakan dengan dinding dan tubuhnya sendiri, serta memilih jalur pergerakan yang efisien menuju makanan. Ini merupakan indikasi bahwa agen telah belajar dari pengalaman sebelumnya dan mampu mengenali skenario yang menguntungkan untuk mendapatkan reward.

Perilaku yang tergambar di sini menandakan bahwa strategi navigasi agen telah berkembang seiring bertambahnya jumlah episode pelatihan. Agen tidak hanya bergerak secara acak, tetapi mulai mengembangkan pola pergerakan yang lebih terstruktur dan aman. Hal ini menunjukkan keberhasilan awal dari proses pembelajaran Reinforcement Learning dalam membentuk perilaku agen yang cerdas dan responsif terhadap kondisi lingkungan permainan.



Gambar 4. 3 Agen berhasil keluar dari jebakan diri sendiri dalam situasi ruang sempit.

Gambar 4.3 ini memperlihatkan salah satu momen penting dalam proses pengujian agen yang telah dilatih. Pada situasi tersebut, agen terjebak dalam pola tubuh yang membentuk ruang sempit menyerupai kurungan. Dalam banyak kasus, kondisi seperti ini akan menyebabkan agen gagal jika tidak memiliki strategi pelarian yang tepat. Namun, agen dalam penelitian ini menunjukkan kemampuan adaptif dengan memilih arah gerak yang tepat untuk keluar dari jebakan tersebut dan melanjutkan permainan.

Perilaku ini menunjukkan bahwa agen telah mempelajari lebih dari sekadar gerakan menuju makanan, ia juga mampu menganalisis kondisi

lingkungannya dan merespons secara strategis terhadap ancaman yang muncul. Kemampuan agen untuk bertahan dalam situasi berisiko tinggi seperti ini menunjukkan bahwa algoritma *Reinforcement Learning* yang diterapkan (khususnya Deep Q-Network) berhasil menginternalisasi skenario kompleks dan menghasilkan pola pengambilan keputusan yang lebih stabil dan aman.

4.2. Evaluasi dan Analisis

Setelah proses implementasi dan pelatihan agen menggunakan algoritma *Deep Q-Network (DQN)* selesai dilakukan, langkah selanjutnya adalah melakukan evaluasi dan analisis terhadap performa agen dalam memainkan permainan Snake. Evaluasi ini bertujuan untuk menilai sejauh mana efektivitas algoritma dalam membentuk perilaku agen yang optimal serta untuk memahami pengaruh berbagai parameter terhadap hasil pelatihan. Analisis dilakukan secara kuantitatif berdasarkan metrik performa seperti skor rata-rata, panjang maksimal ular, dan kestabilan reward, serta secara kualitatif melalui observasi terhadap pola pergerakan dan pengambilan keputusan agen selama permainan berlangsung.

Melalui evaluasi ini, akan diidentifikasi kelebihan dan kekurangan dari pendekatan yang digunakan, serta sejauh mana agen mampu beradaptasi terhadap kondisi permainan yang dinamis. Selain itu, perbandingan antara model *Bellman* dan *Monte Carlo* juga akan dikaji untuk melihat perbedaan performa dan karakteristik pembelajaran masing-masing pendekatan dalam *Reinforcement Learning*.

4.2.1. Analisis *Reward*

Pada penelitian ini, sistem reward dirancang untuk memberikan umpan balik kepada agen atas setiap tindakan yang dilakukan selama permainan Snake berlangsung. Skema reward tersebut menjadi komponen kunci dalam proses pembelajaran, karena berfungsi sebagai sinyal untuk memperkuat perilaku yang mengarah pada keberhasilan serta mengurangi perilaku yang merugikan. Berdasarkan implementasi dalam fungsi play_step() pada file

game.py, agen akan mendapatkan reward positif sebesar +10 setiap kali berhasil memakan makanan, yang merupakan tujuan utama dalam permainan. Sebaliknya, agen akan menerima penalti sebesar –10 jika mengalami kondisi game over, seperti menabrak dinding atau tubuhnya sendiri. Penalti ini bertujuan untuk melatih agen agar menghindari tindakan-tindakan yang mengarah pada kegagalan.

Selain reward utama tersebut, sistem ini juga menerapkan reward tambahan berskala kecil untuk memperhalus proses pembelajaran. Ketika agen bergerak lebih dekat ke arah makanan, maka diberikan reward +0.1, sedangkan ketika bergerak menjauh dari makanan, dikenakan penalti –0.1. Pendekatan ini dimaksudkan agar agen mampu membentuk strategi navigasi yang lebih efisien, bukan hanya berdasarkan hasil akhir, tetapi juga berdasarkan proses menuju tujuan. Di sisi lain, jika agen hanya bergerak tanpa memakan makanan, reward dikurangi sebesar –0.05 untuk mendorong efisiensi dalam pengambilan keputusan dan menghindari gerakan yang sia-sia.

Tabel 4. 1 *Reward Score*

State	Reward	Keterangan
Agen menabrak (menabrak diri sendiri atau dinding)	-10	Penalti besar diperlukan untuk mencegah agen melakukan perilaku buruk
Agen mendekati makanan	0.1	Poin kecil untuk mengarahkan agen menuju makanan
Agen menjauhi makanan	-0.1	Penalti kecil untuk mengarahkan agen menuju makanan
Agen bergerak tetapi tidak mendekati makanan	-0.05	Penalti kecil agar agen dapat melakukan eksplorasi tetapi tidak berputar-putar
Agen memakan makanan	10	Poin besar agar agen terus mengejar goal nya

Penggunaan reward dalam berbagai skala ini memberikan dampak yang signifikan terhadap kualitas pembelajaran agen. Reward positif yang besar mendorong penguatan strategi memperoleh makanan, sedangkan reward bertahap memberikan umpan balik kontinu atas kualitas pergerakan agen. Berdasarkan hasil pelatihan, agen menunjukkan peningkatan performa yang cukup tajam pada 1000 episode pertama dan mampu mempertahankan strategi yang stabil, seperti bergerak mengikuti tepi arena atau menghindari area sempit. Hal ini mencerminkan bahwa sistem reward berhasil mendorong terbentuknya perilaku adaptif yang tidak hanya berorientasi pada tujuan akhir, tetapi juga mempertimbangkan konteks lingkungan dalam setiap langkahnya.

4.2.2. Pengaruh Parameter Terhadap Kinerja

Beberapa parameter penting yang disesuaikan selama pelatihan antara lain:

- *Learning Rate* (α): pengaturan ini mempengaruhi kecepatan agent belajar dari pengalaman.
- *Discount Factor / Gamma* (γ): mempertimbangkan nilai jangka panjang dalam pengambilan keputusan.
- *Exploration Rate / Epsilon* (ε): Awalnya tinggi dan dikurangi secara bertahap hingga mendapatkan batas terenda untuk beralih dari eksplorasi ke eksploitasi.
- *Exploration Decay / Epsilon Decay*: Strategi penurunan nilai epsilon agar agent dapat banyak eksplorasi pada awal permainan dan mulai perlahan beralih ke eksploitasi seiring bertambahnya pengalaman *agent*.

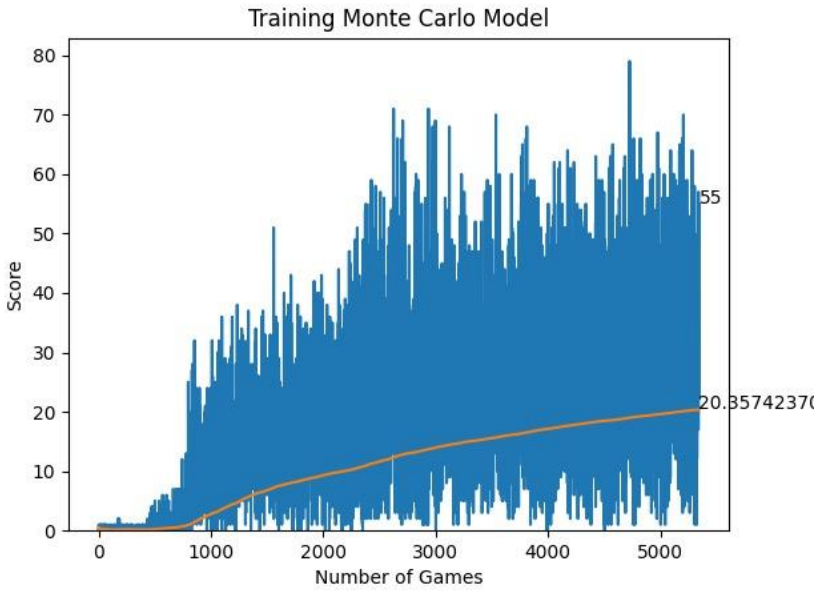
Pengaturan parameter ini berpengaruh signifikan. Misalnya, learning rate yang terlalu tinggi menyebabkan instabilitas reward, sedangkan nilai ε yang tidak diturunkan membuat agent sulit mencapai strategi optimal.

4.3. Perbandingan Kinerja

Reinforcement Learning memiliki banyak metode, beberapa memiliki keunikan dari yang lain. Contohnya saja metode yang akan dibandingkan, yaitu metode *Bellman* dan *Monte Carlo*. Perbandingan ini dilakukan dalam 5000 episode, dengan parameter sebagai berikut:

- Learning rate = 0.001
- Epsilon max = 1.0
- Epsilon min = 0.01
- Epsilon Decay = 0.005
- Discount Rate = 0.9

Pada gambar 4.1 Agent belajar dengan menggunakan metode Bellman, dan Agent belajar dengan cukup baik pada 100 episode pertama, terbukti dengan agen mulai dapat menemukan makanan dan terus mendapatkan reward. Pada episode 100-1000 agent mulai dapat memainkan permainan dengan baik dengan mempelajari strategi untuk menghindari dindin/diri sendiri. Tetapi setelah melewati 1000 episode Agent mendapat nilai yang stagnan, nilai rata-rata Agent memang terus naik tapi tidak ada kenaikan yang signifikan. Sehingga mendapatkan nilai rata-rata akhir di 23.58.



Gambar 4. 4 Grafik Training Model *Monte Carlo* dengan 5000 episode

Pada gambar 4.4 dapat terlihat pada model Monte Carlo untuk 500 episode pertama hasilnya tidak begitu baik. Baru setelah episode 500 – 1500 mendapatkan hasil yang meningkat. Tetapi setelah itu nilai kenaikannya menjadi stabil, tidak ada lagi kenaikan eksponensial. Pada episode 5000 kedua model ini sudah memiliki parameter epsilon 0.01 yang berarti kedua model ini akan terus melakukan eksploitasi model terbaik yang sudah diketahui.

Berdasarkan hasil pelatihan dan evaluasi terhadap dua pendekatan algoritma yang digunakan, yaitu *Bellman* dan *Monte Carlo*, dapat disimpulkan bahwa kedua metode mampu melatih agen untuk memainkan permainan Snake dengan cukup baik. Namun, terdapat beberapa perbedaan signifikan dari segi kecepatan pembelajaran dan stabilitas performa. Pendekatan *Bellman* menunjukkan peningkatan performa yang lebih cepat pada awal pelatihan, khususnya dalam 1000 episode pertama. Agen mampu membentuk strategi dasar dengan cepat, seperti menghindari dinding dan mendekati makanan, sehingga mencapai skor rata-rata sebesar 23,58 dengan panjang ular maksimum 70. Di sisi lain, pendekatan *Monte Carlo*

membutuhkan waktu lebih lama untuk mencapai stabilitas performa. Meskipun hasil awal menunjukkan fluktuasi yang tinggi, agen mulai menunjukkan konsistensi setelah melewati 1500 episode pelatihan. Meskipun secara umum performa akhir dari kedua pendekatan berada pada level yang setara, pendekatan *Bellman* dinilai lebih efisien dari segi waktu pelatihan dan kemampuan awal agen untuk beradaptasi terhadap lingkungan permainan.

Dengan demikian, pendekatan *Bellman* lebih unggul dalam konteks pelatihan agen permainan Snake yang membutuhkan efisiensi dan kestabilan pembelajaran dalam waktu yang relatif lebih singkat. Namun, pendekatan *Monte Carlo* tetap menunjukkan potensi yang baik, khususnya dalam skenario pelatihan jangka panjang dengan ekspektasi hasil yang lebih halus dan akurat.

BAB 5
KESIMPULAN DAN SARAN

Bab ini menyajikan kesimpulan dari hasil penelitian yang telah dilakukan mengenai **"IMPLEMENTASI ARTIFICIAL INTELLIGENCE PADA PERMAINAN SNAKE MENGGUNAKAN METODE *REINFORCEMENT LEARNING"*.** Kesimpulan disusun berdasarkan hasil analisis yang telah dibahas pada bab sebelumnya, mencakup performa agen, efektivitas algoritma, serta perbandingan antara pendekatan *Bellman* dan *Monte Carlo*. Selain itu, bab ini juga memuat saran-saran yang diharapkan dapat menjadi bahan pertimbangan untuk pengembangan penelitian selanjutnya agar lebih optimal.

5.1. Kesimpulan

Berdasarkan hasil penelitian dan analisis yang telah dilakukan, dapat disimpulkan beberapa hal berikut:

1. Implementasi Reinforcement Learning pada permainan Snake berhasil dilakukan dengan baik. Agen AI yang dikembangkan mampu memainkan permainan secara mandiri dan adaptif menggunakan metode Deep Q-Network (DQN). Agen dapat belajar melalui interaksi dengan lingkungan dan menunjukkan peningkatan performa seiring bertambahnya episode pelatihan.
2. Pengaruh parameter pelatihan terhadap performa agen dapat dianalisis secara sistematis. Parameter seperti learning rate, epsilon, discount factor (gamma), serta struktur reward sangat berpengaruh terhadap kualitas pembelajaran agen. Penyesuaian parameter yang tepat terbukti meningkatkan efisiensi dan kestabilan proses pelatihan.
3. Perbandingan antara pendekatan Bellman dan Monte Carlo menunjukkan hasil yang signifikan. Pendekatan Bellman

menunjukkan keunggulan dalam hal kecepatan adaptasi dan kestabilan reward pada fase awal pelatihan, sedangkan pendekatan Monte Carlo membutuhkan waktu lebih lama namun tetap mampu mencapai hasil yang kompetitif pada akhir pelatihan.

5.2. Saran

Berdasarkan hasil penelitian dan evaluasi terhadap implementasi *Reinforcement Learning* dalam permainan Snake, terdapat beberapa saran yang dapat dijadikan acuan untuk penelitian dan pengembangan selanjutnya, yaitu:

1. Melakukan *tuning parameter* yang lebih mendalam terhadap nilai-nilai seperti *learning rate*, discount factor (gamma), dan *exploration rate* (*epsilon*). Meskipun agen berhasil mencapai skor yang stabil, performa cenderung stagnan setelah 1000 episode pertama. Dengan melakukan eksperimen terhadap berbagai kombinasi parameter tersebut, agen kemungkinan dapat mencapai hasil yang lebih optimal serta mengurangi fluktuasi reward.
2. Penggunaan arsitektur jaringan saraf yang lebih kompleks dapat menjadi solusi untuk meningkatkan kemampuan agen dalam mengenali pola lingkungan. Penerapan pendekatan seperti *Double DQN, Dueling DQN*, atau *Prioritized Experience Replay* dapat memperbaiki estimasi nilai Q secara lebih stabil dan akurat, sehingga mempercepat proses pembelajaran serta meningkatkan performa agen secara keseluruhan.
3. Variasi lingkungan permainan perlu dipertimbangkan untuk melatih agen agar mampu beradaptasi di situasi yang berbeda. Penelitian ini masih terbatas pada lingkungan Snake yang relatif statis. Untuk meningkatkan kemampuan generalisasi agen, eksperimen dapat dilakukan pada variasi ukuran grid, kecepatan permainan, atau penempatan makanan secara dinamis, sehingga strategi agen tidak hanya efektif dalam satu kondisi tertentu.

4. Dokumentasi dan visualisasi perilaku agen sebaiknya ditingkatkan melalui penggunaan animasi atau antarmuka visual interaktif. Hal ini akan mempermudah dalam menganalisis strategi yang diambil oleh agen, sekaligus memberikan pemahaman yang lebih baik bagi peneliti dan pembaca terhadap cara kerja algoritma Reinforcement Learning yang diterapkan.

DAFTAR PUSTAKA

Alpaydin, E., 2020. *Introduction to Machine Learning.* 4th ed. Cambridge: MIT Press.

Bostrom, N., 2014. *Superintelligence : paths, dangers, strategies.* 1st ed. Oxford: Oxford : Oxford University Pressv.

Crespo, J. & Wichert, A., 2020. *Reinforcement Learning* applied to games. *SN Applied Sciences,* Volume 2, p. 824.

François-Lavet, V. et al., 2018. An Introduction to Deep. *Foundations and Trends in Machine Learning,* 11(3–4), p. 219–354.

Goodfellow, I., Bengio, Y. & Courtville, A., 2016. *Deep Learning.* Massachusetts: MIT Press.

Harrington, P., 2012. *Machine Learning in Actionv.* 1 ed. New York: Manning Publications.

Kuang, W., 2021. *Fundamentals of Reinforcement Learning.* Edinburg: University of Texas Rio Grande Valley.

Li, Y., 2018. Deep *Reinforcement Learning*. *Clinical Orthopaedics and Related Research,* Volume abs/1810.06339.

Mitchell, T. M., 1997. *Machine Learning.* New York: McGraw-Hill Education.

Mnih, V. et al., 2015. *Human-level control through deep Reinforcement Learning.* s.l.:s.n.

Myerson, R. B., 1991. *Game Theory: Analysis of Conflict.* Cambridge: Harvard University Press.

Russel, S. & Norvig, P., 2020. *Artificial Intelligence A Modern Approach Fourth Edition.* London: Pearson.

Silver, D. et al., 2016. Mastering the game of Go with deep neural networks and tree search. *Nature,* Volume 529, p. 484–489.

Sutton, R. S. & Barto, A. G., 2018. *Reinforcement Learning: An Introduction.* 2nd ed. Cambridge: MIT Press.

Taqwa, A., Isa, I. G. T. & Ariyanti, I., 2023. Designing A WSNs-based Smart Home Monitoring System through. *Rekayasa Sistem dan Teknologi Informasi,* pp. 1224-1232.

Tegmark, M., 2017. *Life 3.0 : being human in the age of artificial intelligence.* New York: Alfred A. Knopf.

LAMPIRAN

LAMPIRAN A :

KODE PROGRAM

A.1. Agent.py

```python
import torch
import random
import numpy as np
from collections import deque
from game import SnakeGameAI, Direction, Point
from model import Linear_QNet, BellmanTrainer
from helper import plot
import time
import sys

MAX_MEMORY = 100_000
BATCH_SIZE = 1000
LEARNING_RATE = 0.001

class Agent:
    def __init__(self, load_model=True):
        self.n_games = 0
        self.epsilon = 0
        self.epsilon_max = 1.0
        self.epsilon_min = 0.01
        # self.epsilon_decay = 0.005
        self.epsilon_decay = 0.01
        self.gamma = 0.9  # discount rate
        self.memory = deque(maxlen=MAX_MEMORY)  # popleft()
        self.model = Linear_QNet(11, 256, 3)
        self.trainer = BellmanTrainer(self.model,
           learning_rate=LEARNING_RATE, gamma=self.gamma)

        # Load model if specified
        if load_model:
            self.load_model()
            print(self.n_games, 'games loaded from
               bellmanModel.pth')
            print(self.epsilon, 'epsilon loaded from
               bellmanModel.pth')

```

```
    def get_state(self, game):
        head = game.snake[0]
        point_l = Point(head.x - 20, head.y)
        point_r = Point(head.x + 20, head.y)
        point_u = Point(head.x, head.y - 20)
        point_d = Point(head.x, head.y + 20)

        dir_l = game.direction == Direction.LEFT
        dir_r = game.direction == Direction.RIGHT
        dir_u = game.direction == Direction.UP
        dir_d = game.direction == Direction.DOWN

        state = [
            (dir_r and game.is_collision(point_r)) or
            (dir_l and game.is_collision(point_l)) or
            (dir_u and game.is_collision(point_u)) or
            (dir_d and game.is_collision(point_d)),

            (dir_u and game.is_collision(point_r)) or
            (dir_d and game.is_collision(point_l)) or
            (dir_l and game.is_collision(point_u)) or
            (dir_r and game.is_collision(point_d)),

            (dir_d and game.is_collision(point_r)) or
            (dir_u and game.is_collision(point_l)) or
            (dir_r and game.is_collision(point_u)) or
            (dir_l and game.is_collision(point_d)),

            dir_l,
            dir_r,
            dir_u,
            dir_d,

            game.food.x < game.head.x,
            game.food.x > game.head.x,
            game.food.y < game.head.y,
            game.food.y > game.head.y
        ]

        return np.array(state, dtype=int)
```

```
76
77      def remember(self, state, action, reward, next_state,
            done):
78          self.memory.append((state, action, reward,
                next_state, done))
79
80      def train_long_memory(self):
81          if len(self.memory) > BATCH_SIZE:
82              mini_sample = random.sample(self.memory,
                    BATCH_SIZE)
83          else:
84              mini_sample = self.memory
85
86          states, actions, rewards, next_states, dones = zip
                (*mini_sample)
87          self.trainer.train_step(states, actions, rewards,
                next_states, dones)
88
89      def train_short_memory(self, state, action, reward,
            next_state, done):
90          self.trainer.train_step(state, action, reward,
                next_state, done)
91
92      def get_action(self, state):
93          self.epsilon = self.epsilon_min + (self.epsilon_max
                - self.epsilon_min) * np.exp(-self.
                epsilon_decay * self.n_games)
94
95          final_move = [0, 0, 0]
96          if random.random() < self.epsilon:
97              move = random.randint(0, 2)
98              final_move[move] = 1
99          else:
100             state0 = torch.tensor(state, dtype=torch.float)
101             prediction = self.model(state0)
102             move = torch.argmax(prediction).item()
103             final_move[move] = 1
104
105         return final_move
106
```

```
107     def save_model(self, filename='bellmanBestModel.pth'):
108         """Save model using trainer's save_model method"""
109         return self.trainer.save_model(filename)
110 
111     def load_model(self, filename='bellmanBestModel.pth'):
112         """Load model using trainer's load_model method"""
113         return self.trainer.load_model(filename)
114 
115 def train(load_previous=True, save_interval=100):
116     plot_scores = []
117     plot_mean_scores = []
118     cumulative_scores = 0
119     record = 0
120     record_on = 0
121     agent = Agent(load_model=load_previous)
122     game = SnakeGameAI()
123     start_time = time.time()
124     longest_time = 0
125     time_on = 0
126 
127     print('Training started...')
128     if load_previous:
129         print('Loading previous model...')
130         agent.load_model('bellmanModel.pth')
131         print(f'Continuing from episode {agent.n_games}')
132     else:
133         print('Starting new training session...')
134         print('Press Ctrl+C to stop training and save
                progress.')
135         print('Training will save checkpoints every',
                save_interval, 'episodes.')
136         print('Use "python agent.py play" to play with the
                trained model.')
137 
138 
139 
140     try:
141         while True:
142             state_old = agent.get_state(game)
143             final_move = agent.get_action(state_old)
144             reward, done, score = game.play_step(final_move
                    )
145             state_new = agent.get_state(game)
146 
147             agent.train_short_memory(state_old, final_move,
                    reward, state_new, done)
148             agent.remember(state_old, final_move, reward,
                    state_new, done)
```

```
149
150             if done:
151                 game.reset()
152                 agent.n_games += 1
153                 agent.train_long_memory()
154                 cumulative_scores += score
155                 running_time = time.time() - start_time
156
157                 if score > record:
158                     record = score
159                     agent.save_model('bellmanBestModel.pth'
                            )
160                     record_on = agent.n_games
161
162                 if agent.n_games % save_interval == 0:
163                     agent.trainer.save_model(f'
                            checkpoint_episode_{agent.n_games}.
                            pth')
164
165                 if running_time > longest_time:
166                     longest_time = running_time
167                     time_on = agent.n_games
168
169                 plot_scores.append(score)
170                 mean_score = cumulative_scores / agent.
                        n_games
171                 plot_mean_scores.append(mean_score)
172                 plot(plot_scores, plot_mean_scores)
173
174                 print(
175                     'Bellman_Learning',
176                     '\nEpisode:', agent.n_games,
177                     '\nScore', score,
178                     '\nHigh_Score:', record, 'pada_episode_
                            ke-', record_on,
179                     '\nAll_Scores:', cumulative_scores,
180                     '\nMean_Score:', mean_score,
181                     '\nEpsilon:', agent.epsilon,
182                     '\nTime:', running_time,
183                     '\nLongest_Time:', longest_time, 'pada_
                            episode_ke-', time_on,
184                     '\n')
185
186                 running_time = 0
187                 start_time = time.time()
188
189     except KeyboardInterrupt:
190         print("Training_interrupted_by_user.")
191         agent.trainer.save_model('bellmanModel.pth')
192         print("Progress_saved!")
```

A.2. Game.py

```
import pygame
import random
from enum import Enum
from collections import namedtuple
import numpy as np

pygame.init()
font = pygame.font.SysFont('arial', 25)

class Direction(Enum):
    RIGHT = 1
    LEFT = 2
    UP = 3
    DOWN = 4

Point = namedtuple('Point', 'x, y')

WHITE = (255, 255, 255)
RED = (200,0,0)
BLUE1 = (0, 0, 255)
BLUE2 = (0, 100, 255)
BLACK = (0,0,0)

BLOCK_SIZE = 20
SPEED = 40

class SnakeGameAI:
    def __init__(self, w=640, h=480):
        self.w = w
        self.h = h
        self.display = pygame.display.set_mode((self.w,
            self.h))
        pygame.display.set_caption('Snake')
        self.clock = pygame.time.Clock()
        self.reset()

    def reset(self):
        self.direction = Direction.RIGHT
```

```
        self.head = Point(self.w/2, self.h/2)
        self.snake = [self.head,
                      Point(self.head.x-BLOCK_SIZE, self.
                          head.y),
                      Point(self.head.x-(2*BLOCK_SIZE),
                          self.head.y)]

        self.score = 0
        self.food = None
        self._place_food()
        self.frame_iteration = 0
```

```
49
50      def _place_food(self):
51          x = random.randint(0, (self.w-BLOCK_SIZE )//
                BLOCK_SIZE )*BLOCK_SIZE
52          y = random.randint(0, (self.h-BLOCK_SIZE )//
                BLOCK_SIZE )*BLOCK_SIZE
53          self.food = Point(x, y)
54          if self.food in self.snake:
55              self._place_food()
56
57
58      def play_step(self, action):
59          self.frame_iteration += 1
60          for event in pygame.event.get():
61              if event.type == pygame.QUIT:
62                  pygame.quit()
63                  quit()
64
65          prev_distance = abs(self.head.x - self.food.x) +
                abs(self.head.y - self.food.y)
66
67          self._move(action)
68          self.snake.insert(0, self.head)
69
70          reward = 0
71          game_over = False
72          if self.is_collision() or self.frame_iteration >
                100 * len(self.snake):
```

```
73              game_over = True
74              reward = -10
75              return reward, game_over, self.score
76
77          new_distance = abs(self.head.x - self.food.x) + abs
                (self.head.y - self.food.y)
78
79          if new_distance < prev_distance:
80              reward += 0.1
81          else:
82              reward -= 0.1
83
84          if self.head == self.food:
85              self.score += 1
86              reward = 10
87              self._place_food()
88          else:
89              self.snake.pop()
90              reward -= 0.05
91
92          self._update_ui()
93          self.clock.tick(SPEED)
94
95          return reward, game_over, self.score
96
```

```
97
98      def is_collision(self, pt=None):
99          if pt is None:
100             pt = self.head
101         if pt.x > self.w - BLOCK_SIZE or pt.x < 0 or pt.y >
                self.h - BLOCK_SIZE or pt.y < 0:
102             return True
103         if pt in self.snake[1:]:
104             return True
105
106         return False
107
108     def _update_ui(self):
109         self.display.fill(BLACK)
110
```

```
111         for i, pt in enumerate(self.snake):
112             if i == 0:
113                 pygame.draw.rect(self.display, WHITE,
                        pygame.Rect(pt.x, pt.y, BLOCK_SIZE,
                        BLOCK_SIZE))  # Hijau
114                 pygame.draw.rect(self.display, WHITE,
                        pygame.Rect(pt.x+4, pt.y+4, 12, 12))
115             else:
116                 pygame.draw.rect(self.display, BLUE1,
                        pygame.Rect(pt.x, pt.y, BLOCK_SIZE,
                        BLOCK_SIZE))
117                 pygame.draw.rect(self.display, BLUE2,
                        pygame.Rect(pt.x+4, pt.y+4, 12, 12))
118
119         pygame.draw.rect(self.display, RED, pygame.Rect(
                self.food.x, self.food.y, BLOCK_SIZE, BLOCK_SIZE
                ))
120
121         text = font.render("Score:_" + str(self.score),
                True, WHITE)
122         self.display.blit(text, [0, 0])
123         pygame.display.flip()
124
125
126     def _move(self, action):
127         clock_wise = [Direction.RIGHT, Direction.DOWN,
                Direction.LEFT, Direction.UP]
128         idx = clock_wise.index(self.direction)
129
130         if np.array_equal(action, [1, 0, 0]):
131             new_dir = clock_wise[idx]
132         elif np.array_equal(action, [0, 1, 0]):
133             next_idx = (idx + 1) % 4
134             new_dir = clock_wise[next_idx]
135         else:
136             next_idx = (idx - 1) % 4
137             new_dir = clock_wise[next_idx]
138
139         self.direction = new_dir
140
```

```
141         x = self.head.x
142         y = self.head.y
143         if self.direction == Direction.RIGHT:
144             x += BLOCK_SIZE
145         elif self.direction == Direction.LEFT:
146             x -= BLOCK_SIZE
147         elif self.direction == Direction.DOWN:
148             y += BLOCK_SIZE
149         elif self.direction == Direction.UP:
150             y -= BLOCK_SIZE
151 
152         self.head = Point(x, y)
```

A.3. Model.py (Bellman)

```python
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import os

class Linear_QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x

    def save(self, file_name='model.pth'):
        model_folder_path = './model'
        if not os.path.exists(model_folder_path):
            os.makedirs(model_folder_path)

        file_name = os.path.join(model_folder_path, file_name)
        torch.save(self.state_dict(), file_name)


class BellmanTrainer:
    def __init__(self, model, learning_rate, gamma):
        self.learning_rate = learning_rate
        self.gamma = gamma
        self.model = model
        self.optimizer = optim.Adam(model.parameters(), lr=self.learning_rate)
        self.criterion = nn.MSELoss()
        self.n_games = 0
        self.epsilon = 1.0
```

```
            os.makedirs(model_folder_path)

        file_path = os.path.join(model_folder_path,
            filename)
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'n_games': self.n_games,
            'epsilon': self.epsilon
        }, file_path)
        print(f"Model saved to {file_path}")

    def load_model(self, filename='bellmanModel.pth'):
        """Load a saved model"""
        model_folder_path = './model'
        file_path = os.path.join(model_folder_path,
            filename)

        if os.path.exists(file_path):
            checkpoint = torch.load(file_path)
            self.model.load_state_dict(checkpoint['
                model_state_dict'])
            print(f"Model loaded from {file_path}")
            print(f"Resumed from episode: {checkpoint['
                n_games']}")
            print(f"Epsilon: {checkpoint['epsilon']}")
            self.n_games = checkpoint['n_games']
            self.epsilon = checkpoint['epsilon']
            return True
        else:
            print(f"No model found at {file_path}")
            return False

    def save_checkpoint(self, filename='bellmanCheckpoint.
        pth'):
        """Save training checkpoint including memory"""
        model_folder_path = './model'
        if not os.path.exists(model_folder_path):
            os.makedirs(model_folder_path)

        file_path = os.path.join(model_folder_path,
```

```
            filename)
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'n_games': self.n_games,
            'epsilon': self.epsilon,
            'memory': list(self.memory)
        }, file_path)
        print(f"Checkpoint saved to {file_path}")

    def load_checkpoint(self, filename='bellmanCheckpoint.
        pth'):
        """Load training checkpoint including memory"""
        model_folder_path = './model'
        file_path = os.path.join(model_folder_path,
            filename)

        if os.path.exists(file_path):
            checkpoint = torch.load(file_path)
            self.model.load_state_dict(checkpoint['
                model_state_dict'])
            self.n_games = checkpoint['n_games']
            self.epsilon = checkpoint['epsilon']

            if 'memory' in checkpoint:
                self.memory = deque(checkpoint['memory'],
                    maxlen=MAX_MEMORY)

            print(f"Checkpoint loaded from {file_path}")
            print(f"Resumed from episode: {self.n_games}")
            return True
        else:
            print(f"No checkpoint found at {file_path}")
            return False
```

```
37          self.epsilon_min = 0.01
38          self.epsilon_decay = 0.005
39
40      def train_step(self, state, action, reward, next_state,
            done):
41          state = torch.tensor(state, dtype=torch.float)
42          next_state = torch.tensor(next_state, dtype=torch.
               float)
43          action = torch.tensor(action, dtype=torch.long)
44          reward = torch.tensor(reward, dtype=torch.float)
45
46          if len(state.shape) == 1:
47              state = torch.unsqueeze(state, 0)
48              next_state = torch.unsqueeze(next_state, 0)
49              action = torch.unsqueeze(action, 0)
50              reward = torch.unsqueeze(reward, 0)
51              done = (done, )
52
53          pred = self.model(state)
54
55          target = pred.clone()
56          for idx in range(len(done)):
57              Q_new = reward[idx]
58              if not done[idx]:
59                  Q_new = reward[idx] + self.gamma * torch.
                       max(self.model(next_state[idx]))
60
61              target[idx][torch.argmax(action[idx]).item()] =
                    Q_new
62
63          self.optimizer.zero_grad()
64          loss = self.criterion(target, pred)
65          loss.backward()
66
67          self.optimizer.step()
68
69      def save_model(self, filename='bellmanModel.pth'):
70          """Save the current model"""
71          model_folder_path = './model'
72          if not os.path.exists(model_folder_path):
```

A.4. Model (Monte Carlo)

```
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import os
import numpy as np

class Linear_QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size
       ):
        super().__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        return self.linear2(x)

    def save(self, file_name='model.pth'):
        model_folder_path = './model'
        if not os.path.exists(model_folder_path):
            os.makedirs(model_folder_path)

        file_name = os.path.join(model_folder_path,
           file_name)
        torch.save(self.state_dict(), file_name)

class MonteCarloTrainer:
    def __init__(self, model, learning_rate, gamma):
        self.learning_rate = learning_rate
        self.gamma = gamma
        self.model = model
        self.optimizer = optim.Adam(model.parameters(), lr=
           self.learning_rate)
        self.criterion = nn.MSELoss()
        self.episode_memory = []
        self.n_games = 0
        self.epsilon = 1.0
```

```
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.005

    def store_experience(self, state, action, reward):
        if isinstance(state, torch.Tensor):
            state = state.detach().numpy()
        elif not isinstance(state, np.ndarray):
            state = np.array(state)

        state = state.flatten()

        self.episode_memory.append({
            'state': state,
            'action': action,
            'reward': reward
        })

    def calculate_returns(self, rewards):
        """Calculate discounted returns using Monte Carlo
           method"""
        returns = []
        G = 0
        for reward in reversed(rewards):
            G = reward + self.gamma * G
            returns.insert(0, G)
        return returns

    def train_episode(self):
        if len(self.episode_memory) == 0:
            return 0

        states = []
        actions = []
        rewards = []

        for experience in self.episode_memory:
            states.append(experience['state'])
            actions.append(experience['action'])
            rewards.append(experience['reward'])

```

```
        returns = self.calculate_returns(rewards)

        try:
            states = torch.tensor(np.array(states), dtype=
               torch.float32)
            actions = torch.tensor(np.array(actions), dtype
               =torch.long)
            returns_tensor = torch.tensor(returns, dtype=
               torch.float32)
        except ValueError as e:
            print(f"Error creating tensors: {e}")
            print(f"States shapes: {[s.shape if hasattr(s,
               'shape') else len(s) for s in states]}")
            self.episode_memory = []
            return 0

        if states.dim() == 1:
            states = states.unsqueeze(0)
        if actions.dim() == 0:
            actions = actions.unsqueeze(0)
        if returns_tensor.dim() == 0:
            returns_tensor = returns_tensor.unsqueeze(0)

        pred_q_values = self.model(states)
        target_q_values = pred_q_values.clone().detach()

        for idx in range(len(actions)):
            action_idx = actions[idx].item()
            target_q_values[idx][action_idx] =
               returns_tensor[idx]

        self.optimizer.zero_grad()
        loss = self.criterion(pred_q_values,
           target_q_values)
        loss.backward()
        self.optimizer.step()

        self.episode_memory = []
        return loss.item()

```

```
        returns = self.calculate_returns(rewards)

        try:
            states = torch.tensor(np.array(states), dtype=
               torch.float32)
            actions = torch.tensor(np.array(actions), dtype
               =torch.long)
            returns_tensor = torch.tensor(returns, dtype=
               torch.float32)
        except ValueError as e:
            print(f"Error creating tensors: {e}")
            print(f"States shapes: {[s.shape if hasattr(s,
               'shape') else len(s) for s in states]}")
            self.episode_memory = []
            return 0

        if states.dim() == 1:
            states = states.unsqueeze(0)
        if actions.dim() == 0:
            actions = actions.unsqueeze(0)
        if returns_tensor.dim() == 0:
            returns_tensor = returns_tensor.unsqueeze(0)

        pred_q_values = self.model(states)
        target_q_values = pred_q_values.clone().detach()

        for idx in range(len(actions)):
            action_idx = actions[idx].item()
            target_q_values[idx][action_idx] =
               returns_tensor[idx]

        self.optimizer.zero_grad()
        loss = self.criterion(pred_q_values,
           target_q_values)
        loss.backward()
        self.optimizer.step()

        self.episode_memory = []
        return loss.item()

```

```
    def train_step(self, state, action, reward, next_state, done):
        """Store experience and train if episode is done"""

        is_batch = isinstance(state, (list, tuple)) or (
            hasattr(state, 'shape') and len(state.shape) > 1
            and state.shape[0] > 1)

        if is_batch:
            states = state if isinstance(state, (list, tuple)) else state
            actions = action if isinstance(action, (list, tuple)) else action
            rewards = reward if isinstance(reward, (list, tuple)) else reward
            dones = done if isinstance(done, (list, tuple)) else [done]

            for i in range(len(states)):
                if isinstance(states[i], torch.Tensor):
                    state_np = states[i].detach().numpy().flatten()
                else:
                    state_np = np.array(states[i]).flatten()

                if isinstance(actions[i], torch.Tensor):
                    if actions[i].dim() > 0 and actions[i].numel() > 1:
                        action_value = torch.argmax(actions[i]).item()
                    else:
                        action_value = actions[i].item()
                else:
                    if hasattr(actions[i], '__len__') and len(actions[i]) > 1:
                        action_value = np.argmax(actions[i])
                    else:
                        action_value = int(actions[i])
```

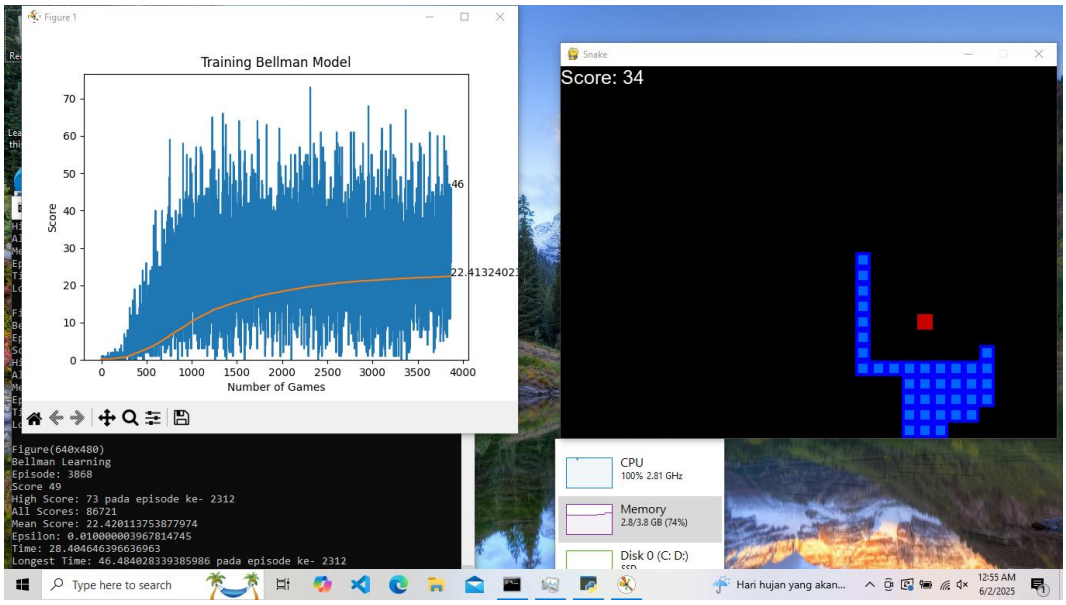
```
                if isinstance(rewards[i], torch.Tensor):
                    reward_value = rewards[i].item() if
                       rewards[i].numel() == 1 else rewards
                       [i].mean().item()
                elif isinstance(rewards[i], (tuple, list)):
                    reward_value = float(rewards[i][0]) if
                       len(rewards[i]) > 0 else 0.0
                else:
                    reward_value = float(rewards[i])

                self.store_experience(state_np,
                   action_value, reward_value)

                if i < len(dones) and dones[i]:
                    return self.train_episode()

            return 0

        else:
            if isinstance(state, torch.Tensor):
                state_np = state.detach().numpy().flatten()
            else:
                state_np = np.array(state).flatten()

            if isinstance(action, torch.Tensor):
                if action.dim() > 0 and action.numel() > 1:
                    action_value = torch.argmax(action).
                       item()
                else:
                    action_value = action.item()
            else:
                if hasattr(action, '__len__') and len(
                   action) > 1:
                    action_value = np.argmax(action)
                else:
                    action_value = int(action)

            if isinstance(reward, torch.Tensor):
                reward_value = reward.item() if reward.
```

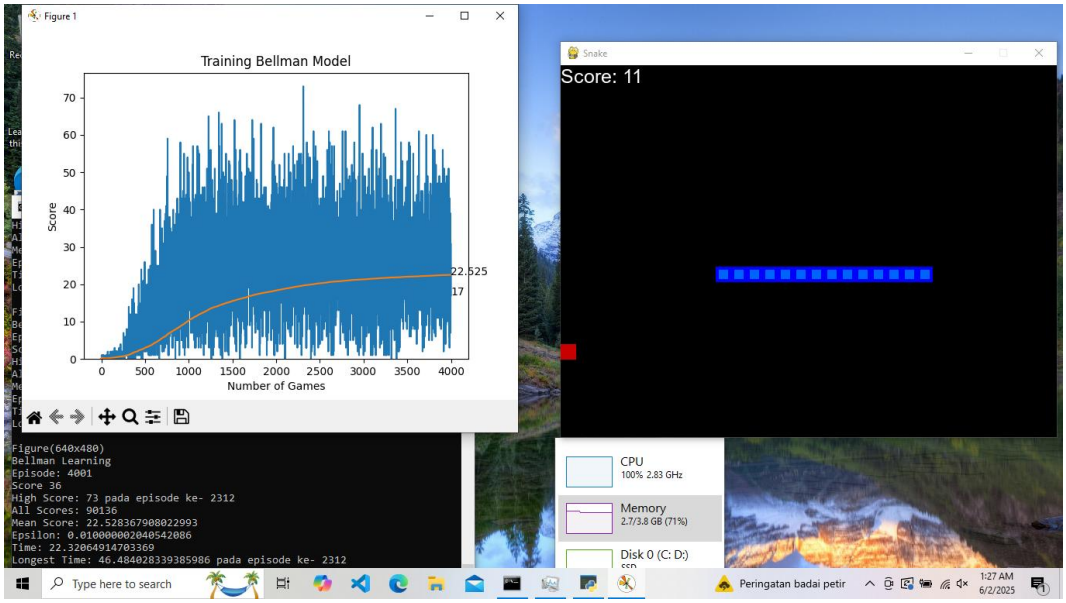
```
                    numel() == 1 else reward.mean().item()
            elif isinstance(reward, (tuple, list)):
                reward_value = float(reward[0]) if len(
                    reward) > 0 else 0.0
            else:
                reward_value = float(reward)

            self.store_experience(state_np, action_value,
                reward_value)

            if done:
                return self.train_episode()

            return 0

    def save_model(self, filename='monteCarloModel.pth'):
        """Save the current model"""
        model_folder_path = './model'
        if not os.path.exists(model_folder_path):
            os.makedirs(model_folder_path)

        file_path = os.path.join(model_folder_path,
            filename)
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'n_games': self.n_games,
            'epsilon': self.epsilon
        }, file_path)
        print(f"Model saved to {file_path}")

    def load_model(self, filename='monteCarloModel.pth'):
        """Load a saved model"""
        model_folder_path = './model'
        file_path = os.path.join(model_folder_path,
            filename)

        if os.path.exists(file_path):
            checkpoint = torch.load(file_path)
            self.model.load_state_dict(checkpoint['
                model_state_dict'])
```

```
            if 'n_games' in checkpoint:
                self.n_games = checkpoint['n_games']
            if 'epsilon' in checkpoint:
                self.epsilon = checkpoint['epsilon']

            print(f"Model loaded from {file_path}")
            print(f"Resumed from episode: {self.n_games}")
            return True
        else:
            print(f"No model found at {file_path}")
            return False

    def save_checkpoint(self, filename='monteCarloCheckpoint.pth'):
        """Save training checkpoint including memory"""
        model_folder_path = './model'
        if not os.path.exists(model_folder_path):
            os.makedirs(model_folder_path)

        file_path = os.path.join(model_folder_path, filename)
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'n_games': self.n_games,
            'epsilon': self.epsilon,
            'memory': list(self.memory)
        }, file_path)
        print(f"Checkpoint saved to {file_path}")

    def load_checkpoint(self, filename='monteCarloCheckpoint.pth'):
        """Load training checkpoint including memory"""
        model_folder_path = './model'
        file_path = os.path.join(model_folder_path, filename)

        if os.path.exists(file_path):
            checkpoint = torch.load(file_path)
            self.model.load_state_dict(checkpoint['model_state_dict'])
            self.n_games = checkpoint['n_games']
            self.epsilon = checkpoint['epsilon']

            if 'memory' in checkpoint:
                self.memory = deque(checkpoint['memory'], maxlen=MAX_MEMORY)

            print(f"Checkpoint loaded from {file_path}")
            print(f"Resumed from episode: {self.n_games}")
            return True
        else:
            print(f"No checkpoint found at {file_path}")
            return False
```

LAMPIRAN B :

DOKUMENTASI VISUAL AGENT





LAMPIRAN C :
RIWAYAT PENULIS

Penulis bernama Sulistyawan Abdillah Rosyid. Penulis lahir di Muara Badak, 19 Mei 1998 dan merupakan anak pertama dari 4 bersaudara. Penulis telah menempuh pendidikan formal di SDN 019 Balikpapan Timur, SMPN 08 Balikpapan, dan SMA Budi Utomo Perak Jombang. Penulis melanjutkan pendidikan tinggi di Institut Teknologi Kalimantan dengan Program Studi Informatika. Selama masa perkuliahan, penulis lebih aktif dalam kegiatan organisasi kepemudaan di luar lingkungan kampus. Penulis pernah menjabat sebagai Staff Ahli Kewirausahaan, dan terlibat aktif dalam berbagai kegiatan sosial dan kepemudaan. Penulis juga telah melaksanakan kerja praktik di Unit Pelaksana Teknis Teknologi Informasi dan Komunikasi (UPT TIK) Institut Teknologi Kalimantan dengan topik pembuatan Sistem Informasi Manajemen Penjaminan Mutu(SIMJAMU). Pada akhir masa studi, penulis mengambil topik Tugas Akhir yang berjudul Implementasi Artificial Intelligence Pada Permainan Snake Dengan Metode Reinforcement Learning.

Biodata Penulis

Sulistyawan Abdillah Rosyid

Muara Badak, 19 Mei 1998

Jl. Mulawarman RT.5 No. 58, Manggar Baru,
Balikpapan Timur, Balikpapan

+62 813 3165 8141

11181079@student.itk.ac.id